

Florian Müller

C64 Tips, Tricks und Tools

Eine Zusammenstellung aller Kniffe rund um den C64
in Basic und Maschinensprache sowie die besten Hilfsprogramme.
Leichtverständliche Erklärungen für Einsteiger
und neueste Anregungen für Insider.

Auf doppelseitig bespielter 5¼"-Diskette (Format 1541) enthalten:
Beispielprogramme und Utilities zum C64, darunter ein Basic-Toolkit,
über 60 Einzeiler sowie Diskettenverwaltungsprogramme.



Commodore Sachbuch



Florian Müller

Tips, Tricks und Tools

Eine Zusammenstellung aller Kniffe
rund um den C64
in Basic und Maschinensprache
sowie die besten Hilfsprogramme

Leichtverständliche Erklärungen für Einsteiger
und neueste Anregungen für Insider

Markt&Technik Verlag AG

CIP-Titelaufnahme der Deutschen Bibliothek

Müller, Florian:
C64 : Tips, Tricks und Tools : e. Zsstellung aller Kniffe rund um d. C64
in Basic u. Maschinensprache sowie d. besten Hilfsprogramme ; leichtverständl. Erklärungen für Einsteiger
u. neueste Anregungen für Insider / Florian Müller.
– Haar bei München : Markt-u.-Technik-Verl., 1988.
(Commodore Sachbuch)
ISBN 3-89090-499-8

15 14 13 12 11 10 9 8 7 6 5
91 90

ISBN 3-89090-499-8

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Jantsch, Günzburg
Printed in Germany

Inhalts-verzeichnis

KAPITEL 1
Tastatur-Tricks oder: Keyboard für Könner

KAPITEL 11
Utilities zum Diskettenlaufwerk

KAPITEL 12
Aktives Anwenden oder: Kniffe zur Software

KAPITEL 13
RAM-Disk-Steuerung oder: Der C576

KAPITEL 14
Der Extended Color Mode oder: Für Farbfans

Anhang:
Die Masterdiskette

Vorwort

Fragt man C64-Besitzer, was ihnen am meistverkauften Computer der Welt im Vergleich zu allen anderen Geräten so gut gefällt, wird man auch folgende Antwort zu hören bekommen: »Bei anderen Computern geht alles nach festgefügten Normen vor sich; die eigene Kreativität kann sich nicht entfalten. Aber beim C64 kommt es darauf an, daß man sich nach und nach in die Materie einarbeitet; dies gibt einem das befriedigende Gefühl, sein Gerät immer mehr zu beherrschen. Vor allem die vielen Tips, Tricks und Hilfsprogramme...«
Lassen Sie mich den letzten Satz fortsetzen:
»...bietet dieses umfassende Buch in einer völlig neuartigen und kompletten Zusammenstellung.«
Keine Frage, besonders die unzähligen Tips, Tricks und Tools machen den C64 so interessant. Das vorliegende Werk möchte Ihnen helfen, das Optimale aus Ihrem Gerät herauszuholen. Dabei ist es in jeder Situation ein hilfreicher Ratgeber und zugleich auch eine Quelle von Informationen und Anregungen. Vor allem Einsteiger möchte es dienlich sein, die vielen kleinen und großen Probleme zu bewältigen, die der C64 aufwirft. Gelingt Ihnen dies erst einmal, so verhilft es Ihnen außer zu regelmäßigen »Aha!«-Erlebnissen und einer steigenden Souveränität auch zu qualifiziertem Wissen über Ihren C64 und das Programmieren im allgemeinen.
Der typische Insider möchte hingegen mit allen erlaubten (und auch unerlaubten) Tricks dem C64 ein Maximum an Leistung entlocken; dafür stellt dieses Werk zahlreiche Programmiertechniken in Basic und Maschinensprache vor, die teilweise in dieser Form noch nie veröffentlicht wurden.
Da die thematische Spannweite dieses Werkes zu groß ist, um in einem Vorwort alle Besonderheiten zu beschreiben, möchte ich Ihnen vorschlagen, einfach loszulegen: Halten Sie die beiliegende Diskette bereit und »here we go«! Auf dem Weg durch die verschiedensten Bereiche des C64 hoffe ich, Sie durch eine bewußt lockere Sprache und hilfreiche Programme für den C64 begeistern zu können. Mein Ziel ist dabei nicht nur, Sie mit fundiertem Wissen zu versorgen, sondern vor allem, Ihnen das zu vermitteln, was wohl unser aller Ziel ist: Spaß am Computern durch sichtbare Erfolge!
Dies versuche ich durch eine Mischung aus neuartigen Techniken sowie einer Zusammenstellung praxisbewährter Hilfsprogramme aus dem 64'er-Magazin zu erreichen. Die beiliegende, doppelseitig randvoll bespielte Programmdiskette böte sogar – ohne Übertreibung – genügend Stoff für mehrere Bücher.

Zur Listing-Lawine

Einen der größten Vorteile dieses Werkes sehe ich in der einzigartigen Programmdiskette (doppelseitig bespielt), die neben unzähligen Utilities auch alle Programmierbeispiele enthält. Wenn Sie dieses Buch durchblättern, werden Ihnen die vielen Listings sicher auffallen; diese waren eine logische Konsequenz aus dem Reichtum an Programmiertricks und fertig verwendbaren Modulen. Ein Listing sagt mehr als tausend Worte!

Zur Entstehung des Buches

Dieses Buch ist zwar in einer recht kurzen Zeitspanne verfaßt worden, doch darin spiegelt sich die Erfahrung aus dreieinhalb Jahren intensiver C64-Begeisterung wider. In der Tat stammen einige der hier gezeigten Tricks aus den ersten Wochen meiner Beschäftigung mit dem C64, doch gerade durch das späte Verfassen des Buches (Ende 1987) ist eine größere Aktualität als bei allen anderen Werken zu diesem Thema gesichert. Zu meiner persönlichen Motivation: Das Thema »Tips, Tricks, Utilities« hat mich schon immer begeistert; es ist kein Zufall, daß mein erstes Buch »Vom C64 zum C128 – Tips und Tricks« hieß, aus dem ich übrigens einige C128-Programmiertechniken für dieses Werk auf den C64 übertragen habe.
Ebenfalls eine wichtige Voraussetzung für die Entstehung dieses Buches war meine Arbeit an »C64 für Insider«; nachdem ich alle internen Zusammenhänge des C64 analysiert hatte, wollte ich diese auch in der Programmierpraxis anwenden und einem breiten Publikum zugänglich machen. Da es jedoch in diesem Buch oftmals zu weit führen würde, den absoluten Profis und denen, die es werden wollen, zu allen Programmiertricks eine völlig detaillierte Beschreibung der Funktionsweise zu liefern, bitte ich um Verständnis, daß ich des öfteren auf »C64 für Insider« hinweise. Es ist auch in Ihrem Interesse, wenn ich eine Fundgrube an Know-how für (angehende) Profis nenne, die wohl die bestmögliche Weiterführung oder Ergänzung dieses Buches darstellt.

Danksagungen

Trotz der optimalen Basis bin ich vor allem drei Personen zu größtem Dank verpflichtet: Frank Hergenröder, mein Lektor beim Markt&Technik-Verlag, schlug mir nicht nur dieses interessante Thema vor und versorgte mich mit wichtigem Material, sondern zeigte auch viel Geduld, als dieses Projekt durch anderweitige Zwischenarbeiten (GEOS läßt grüßen!) mehrfach aufgehalten wurde.
Dietrich Weineck, Autor zahlreicher Programme des 64'er-Magazins (SMON, MSE, Super Copy), hat Ihnen wie mir mit der Freigabe des Quelltextes zu »Super Copy« eine große Freude gemacht; sein persönliches Entgegenkommen sei hier gewürdigt.
Thomas Fenzl hat mir nun bereits zum dritten Mal als fachlicher und sprachlicher Korrektor großen Beistand gewährt; ohne seine kritischen Anmerkungen, Verbesserungsvorschläge und seine »flexible Arbeitszeit« hätte ich mich schwer getan.

Florian Müller, München

1 Tastatur-Tricks oder: Keyboard für Könner

Obwohl in letzter Zeit immer neuere Eingabemedien erfunden wurden (man denke nur an die »Maus-Welle«, die mit GEOS auch den C64 erfaßt, wenn nicht sogar überrollt hat), bleibt die Tastatur ein unersetzlicher Grundbestandteil aller Computersysteme. Die Erfahrenheit eines Computeranwenders zeigt sich bereits in der Bedienung der Tasten: Der Profi flitzt nur so über die Tastatur, der Anfänger bedient sich mit Mühe des »Adler-Suchsystems«.
Deshalb kommt es gerade darauf an, die C64-Tastatur möglichst gut zu beherrschen. Sobald Sie deren vielfältige Möglichkeiten bei der C64-Bedienung sowie in eigenen Programmen nutzen lernen, stellt sich automatisch mehr Erfolg und Freude am C64 ein. Dieses Kapitel vermittelt Ihnen grundlegende Kenntnisse, weiht aber auch die ausgefuchsten Profis in raffinierte und effektive Programmiertechniken für Basic und Assembler ein.

1.1 Hilfen zum Editor

Über den angeblich spartanischen Editor des C64 hat sich schon mancher Fachautor den Mund zerrissen. Dennoch sei hier eine Lanze für ihn gebrochen, der zwar nicht an vollwertige Textverarbeitungssysteme heranreicht, jedoch immerhin einen praktischen »Full-Screen-Editor« darstellt; im Gegensatz dazu können CP/M- oder MS-DOS-Anwender ein Lied davon singen, wie ihnen der primitive Zeileneditor des Betriebssystems tagtäglich zu schaffen macht...
Deshalb lautet die Devise dieses Unterkapitels: Reizen wir die Fähigkeiten des Full-Screen-Editors zielstrebig aus!

1.1.1 Automatisches Laden und Starten

Selbst Floppy-Speedersysteme verfügen nicht unbedingt über eine nützliche, weil Tipparbeit sparende Funktion: das automatische Laden und Starten eines Programms mit nur einer (!) Befehlseingabe.
Möchte man ein Programm laden und starten, benötigt man also im Regelfall zwei nacheinander einzugebende Befehle:

```
LOAD "PROGRAMM",8        <RETURN>
RUN                      <RETURN>
```

Die Eingabe von

```
LOAD "PROGRAMM",8:RUN   <RETURN>
```

mag zwar verlockend erscheinen, scheitert aber daran, daß alle Befehle hinter der LOAD-Anweisung geflissentlich ignoriert werden...
Es gibt jedoch einen recht simplen, aber wirkungsvollen Ausweg. Geben Sie zunächst die Ladeanweisung ein, ohne sofort <RETURN> zu drücken:

```
LOAD "PROGRAMM",8:
```

Nun kommt der eigentliche Trick bei der ganzen Sache zur Geltung: Lösen Sie <SHIFT> und <RUN/STOP> gleichzeitig aus. Das Ergebnis kann sich sehen lassen:

```
LOAD "PROGRAMM",8:LOAD

SEARCHING FOR PROGRAMM
LOADING
READY.
RUN
```

Das Programm wird also in einem Durchgang geladen und gestartet. Sie haben nicht nur einige Tipparbeit gespart, sondern dürfen sich nun dank des selbsttätigen Starts bequem zurücklehnen, bis Sie das Titelbild des Programms wieder an den Computer zurückholt.

Wie funktioniert das?
Wenn Sie nun einen Ausflug in die Regionen der Maschinenprogrammierung befürchten, so glauben Sie mir: Die Funktionsweise ist auch für Einsteiger durchaus einleuchtend. Man muß nämlich nur bedenken, daß die Tastenkombination <SHIFT> + <RUN/STOP> nichts anderes ist als das aufeinanderfolgende Drücken folgender Tasten:

```
<l><o><a><d><RETURN><r><u><n><RETURN>
```

Beim ersten RETURN versucht der C64 folgende Zeile auszuführen:

```
LOAD "PROGRAMM",8:LOAD
```

Das erste LOAD stammte aus unserer Von-Hand-Eingabe, das zweite bereits ist auf <SHIFT> + <RUN/STOP> zurückzuführen. Erinnern Sie sich noch? Der C64 »vergißt« bei LOAD-Befehlen ungeprüfterweise alles, was hinter dem Doppelpunkt steht. Das zweite LOAD ist also völlig irrelevant, uns kommt es nur auf das <RETURN> an. Nach dem Laden des Programms ist der C64 dann wieder für Eingaben aufgeschlossen. Bitte sehr, da warten noch weitere Tasten aus dem SHIFT-RUN/STOP-Text auf ihre Berücksichtigung: Die Zeichen

```
RUN <RETURN>
```

werden also nicht anders aufgefaßt, als wenn Sie es Buchstabe für Buchstabe eingetippt hätten, und dieser Befehl löst bekanntlich einen Programmstart aus.

1.1.2 RUN mit zwei Tastendrücken

Hat man sich einmal mit dem in 1.1.1 vorgestellten Trick vertraut gemacht, fällt es auch nicht mehr schwer, ihn für eine weitere Tipp-Erleichterung einzusetzen. Diesmal wollen wir unter der Voraussetzung, daß sich bereits ein Programm im Speicher befindet, dieses mit möglichst wenigen Tastendrücken starten. Am günstigsten ist natürlich eine Funktionstastenbelegung, wie sie bei Floppy-Speedern in der Regel besteht; es geht aber auch anders:

```
K<SHIFT>+<RUN/STOP>
```

hat dieselbe Wirkung, wenn auch dabei ein chaotischer Bildschirmaufbau entsteht:

```
KLOAD
?SYNTAX ERROR
READY.
RUN
```

Wiederum wird der LOAD-Befehl ignoriert, da er in Verbindung mit K (Sie können wahlweise auch einen anderen Buchstaben verwenden) eine syntaktisch falsche Anweisung bildet (es gibt nur LOAD, nicht aber KLOAD o.ä.). Auf den SYNTAX ERROR hin wird wiederum RUN <RETURN> simuliert.

1.1.3 CURSOR LEFT effektiv verwendet

Beim Editieren am Bildschirm gerät man permanent über den rechten Rand und schreibt somit in der nächsten Zeile weiter. Umgekehrt ist es aber auch möglich, von der ersten (am weitesten links stehenden) Spalte einer Zeile in die letzte (am weitesten rechts stehende) der vorausgegangenen Zeile zurückzuspringen.
An einem Beispiel läßt sich verdeutlichen, wie nützlich dies ist. Nehmen wir an, folgende beiden Zeilen stehen am Bildschirm

```
10 PRINT "NUR EIN BEISPIEL      (DEMo)"
20 PRINT "================      ======
```

Der Cursor befindet sich nun unterhalb der »2« von »20«. Sie stellen gerade mit mittlerem Entsetzen fest, daß das »o« am Ende von Zeile 10 sträflich klein geschrieben ist. Was tun, um den Flüchtigkeitsfehler auszubessern?
Normalerweise würde man zunächst den Cursor zweimal nach oben bewegen (CRSR UP) und dann solange auf <CRSR RIGHT> drücken, bis das »o« erreicht ist. Viel schneller und schonender ist jedoch folgende Alternative:

- Drücken Sie ein einziges Mal auf <CRSR UP>, woraufhin der Cursor auf der »2« steht.
- Lösen Sie nun solange bzw. so oft <CRSR LEFT> aus, bis der Cursor augenblicklich auf »o« landet.

Nach dem ersten <CRSR LEFT> steht er bereits am rechten Ende der Bildschirmzeile, in welcher Zeile 10 zu sehen ist. Von dort ist es nur ein Katzensprung zum »o«.

1.1.4 HOME, sweet HOME

Beobachtet man durchschnittliche C64-Anwender, so verkennen sie – so belanglos das auch klingen mag – den großen Nutzeffekt einer Taste: <HOME>
Dabei ist gerade diese eine echte Arbeitserleichterung. Ein kleines Beispiel soll Ihnen als Anregung dienen, auch einmal an die HOME-Taste zu denken, die viele Cursorbewegungen stark beschleunigt:
Nehmen wir an, in der obersten Bildschirmzeile stand ein Befehl wie LOAD "$",8. Nach seiner Ausführung und anschließendem LIST ist der Cursor bereits in einer der letzten Bildschirmzeilen. Da Sie jedoch im angezeigten Directory nicht fündig geworden sind, wollen Sie den Befehl LOAD "$",8 erneut ausführen und tragen sich deshalb mit dem Gedanken, durch etwa zwanzigfaches <CURSOR UP> wieder in die entsprechende Bildschirmzeile zu springen, um den Befehl nur noch mit <RETURN> zu bestätigen – anstatt ihn noch einmal komplett neu einzugeben. Sicherlich kommen einige auch darauf, lieber gleich <HOME> zu drücken. Würden Sie daran aber auch dann denken, wenn der Befehl LOAD "$",8 in der zweiten, dritten oder vierten Bildschirmzeile steht?
Selbst dann erweist es sich nämlich als vorteilhaft, zuerst <HOME> und dann für kurze Zeit <CURSOR DOWN> gedrückt zu halten, anstatt solange zu warten, bis unzählige Male <CURSOR UP> ausgeführt wurde.
Für die ganz Genauen sei noch gesagt, daß <HOME> sogar um mehrere Millionstel Sekunden schneller (!) ist als jede andere Cursorbewegung, und zwar in völliger Unabhängigkeit von der Bildschirmposition vor der HOME-Bewegung. Ich war selbst verblüfft, als ich die entsprechenden Bestandteile des Betriebssystems analysierte. Interessierte verweise ich daher auf das Gesamtresultat meiner Dokumentationsbemühungen, das Buch »C64 für Insider«.
Ihnen allen gebe ich jedoch, sowohl für die Tastaturbedienung als auch die Programmierung von Bildschirmausgaben, einen Rat auf den Weg: Let's go HOME!

1.1.5 Die Wanzen in der Tastatur

Programmfehler werden mit dem Fachbegriff »bug«, also »Wanze« bezeichnet. Wo sich diese niedlichen Tierchen in der C64-Tastatur eingenistet zu haben scheinen, erfahren Sie hier klipp und klar.

- Halten Sie die rechte SHIFT-Taste gedrückt und versuchen Sie, gleichzeitig auf <,> und <.> mit gewisser Geschwindigkeit einzuhämmern. Am Bildschirm könnte folgendes Ergebnis erscheinen:

  <?><?><?>><<?><?>><?><??><?>

Noch erstaunlicher ist, daß dies auch für folgende Tasten gilt, wenn <SHIFT> gedrückt bleibt: <X> und <C>, <V> und <B>, <N> und <M>.
Fragt sich natürlich, wo das Fragezeichen herkommt, das doch nicht auf den gedrückten Tasten liegt. Ich erlaube mir, die Frage an die C64-Entwickler weiterzugeben... Ganz ehrlich: Im Betriebssystem konnte ich den verantwortlichen Schwachpunkt trotz intensiver Bemühungen nicht lokalisieren. Wahrscheinlich ist das Kernel schlicht und einfach durch drei gleichzeitig ausgelöste Tasten überfordert.

- Auch die linke SHIFT-Taste eignet sich für einen derartigen Trick. Um dasselbe wie <SHIFT> + <A> zu erreichen, bietet sich – man glaubt es kaum – auch folgende Dreierkombination an: linkes <SHIFT> + <CRSR UP/DOWN> + <CRSR RIGHT/LEFT>
- Mit dem dritten und letzten »Tripel« wollen wir es gut sein lassen:
  <C=> + <;> + <=> entspricht <CTRL> + <7>
- Einen echten »Hammer« habe ich mir bis zum Schluß aufgespart. Er funktioniert jedoch nur auf sehr alten C64-Modellen, dort jedoch mit verheerender Wirkung. Dafür ist sein Zustandebringen etwas komplizierter. Fahren Sie dazu mit dem Cursor in die unterste Bildschirmzeile und drücken Sie genau 80mal auf <*>. Nun bringt <DEL> einen mit dem entsprechenden Fehler behafteten C64 aus der Fassung:

```
LOAD
?SYNTAX ERROR
READY.
RUN
READY.
```

So weit, so gut. Die Krone setzt dem Ganzen noch <SHIFT> + <3> auf, woraufhin ungewollterweise ein Ladevorgang von Kassette erfolgt.
Es sei noch einmal darauf hingewiesen, daß dieser Fehler nur auf den allerältesten C64-Exemplaren besteht. Dem Pioniergeist der damaligen C64-Käufer hat er jedoch keinen Abbruch getan!

- Auf jedem C64 (sofern kein verändertes ROM, wie bei Floppy-Speedern, vorliegt) läßt sich ein ähnlich chaotischer Zustand durch einen SYS-Befehl hervorrufen, auf den mich zugegebenermaßen Kommissar Zufall gebracht hat:

```
SYS 62391
```

Danach wird jede mit <RETURN> bestätigte Eingabe, so richtig sie auch sein mag, mit »?SYNTAX ERROR« quittiert. Durch Betätigen der Tastenkombination <SHIFT> + <CLR/HOME> wird es sogar noch schlimmer. Der Cursor flackert hektisch in der linken oberen Ecke des Bildschirms. Nur noch einige wenige Tasten, zum Beispiel <E>, sind überhaupt noch ansprechbar. Ansonsten stellt der Computer sich »tot«. Dies ist ganz interessant,

wenn man zu Kopierschutzzwecken ein Programm auf derartige Weise beenden möchte. Dazu verwendet man eine Zeile des folgenden Inhalts:

```
100 SYS 62391:X
```

Der nicht vorhandene Befehl X löst den SYNTAX ERROR und damit das Programmende aus. Probieren Sie es aus, aber Vorsicht!: Das im Speicher befindliche Programm geht verloren.

1.1.6 CTRL-Simulationen von Steuertasten

Die Steuertasten und die Zeichen, die von ihnen bewirkt werden, nämlich die sogenannten Steuerzeichen, sind eigentlich Thema von 1.2. Hier seien jedoch schon Tastenkombinationen vorgestellt, mit denen sich andere Tasten simulieren lassen. Dies ist zum einen ein interessantes, zum Experimentieren anregendes Feld, zum anderen oftmals eine Erleichterung. So ist <CTRL> + <E> für manchen Anwender leichter zu drücken als <CTRL> + <2>. Folgende CTRL-Simulationen gibt es:

<CTRL> + <E>	entspricht <CTRL> + <2>
<CTRL> + <H>	entspricht PRINT CHR$(8);
<CTRL> + <I>	entspricht PRINT CHR$(9);
<CTRL> + <M>	entspricht <RETURN>
<CTRL> + <N>	entspricht PRINT CHR$(14);
<CTRL> + <Q>	entspricht <CURSOR DOWN>
<CTRL> + <R>	entspricht <RVS ON>
<CTRL> + <S>	entspricht <HOME>
<CTRL> + <T>	entspricht <DELETE>
<CTRL> + <;>	entspricht <CURSOR RIGHT>
<CTRL> + <£>	entspricht PRINT CHR$(28);
<CTRL> + <↑>	entspricht PRINT CHR$(30);
<CTRL> + <←>	entspricht PRINT CHR$(31);

Probieren Sie alle diese Tastenkombinationen einmal aus. Aus der Reihe fallen dabei H, I und N; diese CTRL-Simulationen sind nicht als Steuertasten vorhanden, zeigen aber dennoch interessante Wirkung:

<CTRL> + <H>	blockiert die Tastenkombination <SHIFT> + <C=>.
<CTRL> + <I>	hebt die Blockade durch <CTRL> + <H> auf.
<CTRL> + <N>	schaltet, unabhängig von einer eventuellen Sperre für <SHIFT> + <C=>, auf den Kleinschriftmodus um.

Aus der vorausgegangenen Tabelle geht noch etwas hervor: Nicht alle Steuertasten können über <CTRL> vorgetäuscht werden. Dies sollten Sie bis 1.2 im Hinterkopf behalten.

1.1.7 Tastatur-Simulation mit Joystick – und umgekehrt!

Eigentlich sollte man meinen, daß Joystick und Tastatur voneinander völlig unabhängig sind; dem ist jedoch nicht so. Dies ist Ihnen vielleicht schon aufgefallen, wenn Sie in Port 1 einen Joystick hatten und diesen bewegten: Am Bildschirm erscheinen die unterschiedlichsten Zeichen. Bei Dauerfeuer prasseln vor allem die Leerzeichen nur so nieder, da der Feuerknopf des Joysticks in Port 1 vom C64 mit der Leertaste verwechselt wird. Auch mit einem Joystick in Port 2 ergeben sich unerwartete Effekte, wenn auch nicht so offensichtlich wie in Port 1. Halten Sie einmal die Taste <N> gedrückt und lösen Sie den Feuerknopf Ihres Port-2-Joysticks aus: Das Ergebnis unterscheidet sich nicht von <SHIFT> + <RUN/STOP>. In Verbindung mit dem in 1.1.2 vorgestellten Verfahren läßt sich damit eine »Optimierung« der RUN-Auslösung auf nur noch einen einzigen Tastendruck erzielen: Drücken Sie einmal <N>, so daß dieses Zeichen am Bildschirm erscheint, und lösen Sie bei (noch) gedrücktem <N> den Feuerknopf aus!

Auch in Verbindung mit anderen Tasten wirken sich Joystickbewegungen in Port 2 aus; solange allerdings keine Taste gedrückt wird, gibt sich der Joystick sehr verträglich.

Selbstverständlich liegt der Schluß nahe, daß auch der Joystick durch Tastendrücke simuliert werden kann. So ist es auch; folgende Tastendrücke simulieren den Joystick in Port 1:

SPACE	Feuer
CTRL	links
1	oben
2	rechts
<	unten

So wird Joystick 2 überflüssig:

<CTRL> + <J>	Feuer
<CTRL> + <D>	links
<CTRL> + <CURSOR RIGHT>	oben
<CTRL> + <G>	rechts
<CTRL> + <A>	unten

Natürlich muß man schon ein richtiger Fingerakrobat sein, um diese Tastenkombinationen anwenden zu können. Außerdem sind gute Programme darauf vorbereitet und fangen solche Tastendrücke ab. Dennoch ist es damit für Nur-Tastaturler z.B. möglich, ein Programm anzusehen, das eigentlich einen Joystick erfordert.

Umgekehrt können Sie den Joystick in Port 2 unter Zuhilfenahme des JOYCURSOR, eines kleinen Utilitys auf der Masterdiskette zu diesem Buch, dazu verwenden, die Cursortasten durch Joystickbewegungen wahlweise zu ersetzen:

```
LOAD "JOYCURSOR $C000",8,1
NEW
SYS 49152
```

Die Funktionsweise dieses Programms beschreibt 1.3.

1.1.8 Verbotene Variablen trotzdem verwenden

Definitionsgemäß sind Computer durch ihre Logik gekennzeichnet. Doch wo bleibt diese Logik, wenn folgende – auf den ersten Blick korrekte – sinnvolle Eingabe mit einem lapidaren SYNTAX ERROR verschmäht wird:

```
TO=5
```

Schließlich ist TO unter Umständen ein sinnvoller Variablenname; warum also akzeptiert ihn der Basic-Interpreter nicht?

Ein Blick ins Handbuch, Überschrift »Reservierte Schlüsselwörter«, zeigt, daß sämtliche Basic-Schlüsselwörter nicht in Variablennamen verwendbar sind; darunter fällt auch der TO-Befehl aus Anweisungen wie FOR X=1 TO 1000. Gleiches gilt auch für IF und FN. Verwendet man, was durchaus erlaubt ist, längere Variablennamen, läuft man permanent Gefahr, irgendein Schlüsselwort unbeabsichtigt einzubauen. Überlegen Sie selbst, warum folgende Variablennamen unzulässig sind:

ROTOR	(enthält TO)
WINTER	(enthält INT)
GOOD	(enthält GO)
VORHER	(enthält OR)

Sie sehen, daß unglaublich viele sinntragende Variablennamen durch die Schlüsselwörter-Sperre verhindert werden. Beim C128 mit seinen noch zahlreicheren »Keywords« ist dies sogar schon uferlos, da selbst »DO« als SYNTAX ERROR identifiziert wird. Lassen wir uns das gefallen? Natürlich nicht, denn mit einem einfachen Trick haben wir wieder freie Hand bei der Wahl der Variablennamen. Um auf das Beispiel TO=5 zurückzukommen, geben Sie doch einmal folgendes ein:

```
T<SHIFT>+<SPACE>0=5
```

Am Bildschirm erscheint dabei

```
T 0=5
```

Es entsteht allerdings – ein deutlicher Fortschritt – keine Fehlermeldung mehr. Nun interessiert uns auch, ob die Variable in der verwendeten Schreibweise tatsächlich vermerkt wurde:

```
PRINT T<SHIFT>+<SPACE>0
```

liefert das korrekte Ergebnis 5. Bemerkenswert dabei ist die Tatsache, daß das <SHIFT> + <SPACE> zwar über Tastatur als Abgrenzungsmarke einzugeben bleibt, aber der eigentliche Variablenname nur aus T und O besteht – ohne daß dies mit dem Basic-Befehl TO verwechselt wird. Zur Verdeutlichung wollen wir die beiden angegebenen Befehle in eine Beispiel-Programmzeile integrieren:

```
1 T<SHIFT>+<SPACE>0=5:?T<SHIFT>+<SPACE>0
```

ergibt am Bildschirm folgende Zeile:

```
1 T O=5:?T O
```

Nach LIST sehen wir das überzeugende Ergebnis (das SHIFT-SPACE verschwindet, der Variablenname besteht fort):

```
1 TO=5:PRINTTO
```

Die Ausführung über RUN ist der letztliche Beweis für die volle Funktionsfähigkeit dieses Tricks. Man kann also feststellen, daß das <SHIFT SPACE> lediglich bei der Befehlseingabe von Bedeutung ist; es ist im Grunde ein bedeutungsloses Zeichen (deshalb verschwindet es nach der Eingabe automatisch), verhindert aber zunächst, daß der C64 die Zeichen TO des Variablennamen als Basic-Befehl mißversteht. Anstelle von <SHIFT> + <SPACE> dürfen Sie auch <SHIFT X> oder andere Kombinationen <SHIFT> + <Buchstabe> verwenden. Hier noch für ausgesprochene Insider die entsprechende Beweisführung, die sich anhand eines kommentierten ROM-Listings und Systemhandbuches (Commodore-Sachbuch »C64 für Insider«, Markt&Technik-Verlag) vorzüglich nachvollziehen läßt:

- Die Tokenisierungsschleife ist bestrebt, alle überflüssigen – also nicht zuzuordnenden – Zeichen wie <SHIFT> + <SPACE> zu entfernen. Deshalb erscheinen sie bei späterem LIST nicht mehr.
- Vor dieser Eliminierung erfolgt jedoch die Umwandlung aller Schlüsselwörter in Tokens. Dabei wird ein einfacher Vergleichsalgorithmus verwendet, der nur bei eindeutiger Erkennung eines Schlüsselwortes das Token bildet. Ein mögliches <SHIFT> + <SPACE> täuscht somit einen »normalen« Variablennamen vor.
- Der spätere SYNTAX ERROR entsteht nur, weil der Variablenname TO (ohne zwischengestelltes SHIFT-SPACE) in das TO-Token $a4 umgewandelt wird. Dieses wiederum ist kein zulässiger Variablenname im Sinne des Basic-Interpreters (für den ein Variablenname mit einem Buchstabencode, also $41-$5a, zu beginnen hat), so daß er diesen schließlich zurückweist.
- Steht jedoch die untokenisierte Bezeichnung TO als ASCII-Code $54 $4f im Speicher, so ist dies für den Basic-Interpreter nichts Fehlerhaftes; die Tokenisierung erfolgt schließlich nur einmal – bei der Befehlseingabe –, später jedoch entfällt dieser zeitaufwendige Bearbeitungsschritt.

1.2 Steuerzeichen

Spätestens wenn Sie einmal versucht haben, Ihr Textprogramm an Ihren Drucker anzupassen, sind Sie auf den Begriff »Steuerzeichen« gestoßen. Aber auch beim Programmieren erweisen sie sich als nützliche Hilfsmittel.

1.2.1 Vorteile und Möglichkeiten von Steuerzeichen

Betrachtet man die Textausgabefähigkeiten des C64 auch im Vergleich mit anderen 8-Bit-Computern (Atari XL/XE, Sinclair ZX, Apple II), so fällt neben der Vielzahl von Grafikzeichen vor allem ein Merkmal ins Auge: Der Full-Screen-Editor ist bereit, jedes beliebige Steuerzeichen zu jeder Zeit auszuführen. Der Vorteil liegt darin, daß Ihnen ein mächtiges Instrumentarium für die Bildschirm-Bearbeitung zur Verfügung steht, sei es beim Editieren oder beim Programmieren. Ein Beispiel: Andere Computer repräsentieren die Funktion »Bildschirm löschen« durch Befehle wie CLS; beim C64 geschieht dasselbe bei Ausgabe des Zeichencodes 147 ($93). Beherrscht man einmal die vorhandenen Steuerzeichen (wozu eine Tabelle im Handbuch enthalten ist), bieten diese folgende unschätzbaren Vorteile:

- Steuerfunktionen sind auch in Strings zwischenzuspeichern; dadurch lassen sich
  Textattribute unmittelbar in Zeichenketten einbinden,
  längere Steuersequenzen auf Abruf wiederholen.
- Die Ansteuerung von Druckern vollzieht sich ebenfalls durch Steuerzeichen. Für einen C64-Programmierer sind diese bereits von der Bildschirmausgabe her eine Selbstverständlichkeit; zudem entsprechen sich manche Steuerzeichen im ASCII-Code (Beispiel: Carriage Return = Wagenrücklauf).
- Steuerzeichen zeigen mit einigen Tricks auch während Eingaben in/von aktuellen Programmen ihre Wirkung. Der Anwender hat damit direkten Einfluß auf die Bildschirmgestaltung eines laufenden/entstehenden Programms.
- Programme, die Steuerzeichen in Ausgabestrings enthalten, sind für den eingermaßen geübten Programmierer viel schneller zu überblicken als solche, die sich für alle Steuerfunktionen unterschiedlicher Basic-Befehle bedienen. Beispiel: Das reverse Herz (oder reverse große »S« im Kleinschriftmodus) erkennt man rasch als »Bildschirm löschen«, vor allem im eindeutigen Zusammenhang mit der PRINT-Anweisung.

1.2.2 Die Eingabemodi

Wenn Sie die Tastatur Ihres C64 bedienen, denken Sie dann auch daran, »was eigentlich dahinter steckt«? Auf jeden Fall ist es nicht nur interessant, sondern auch sehr nutzbringend, einmal hinter die Kulissen des Editors zu sehen – soweit es auch ohne Maschinensprache-Kenntnisse zu verstehen ist. Tiefschürfendere Profi-Informationen finden Sie an anderer Stelle (»C64 für Insider«, worauf ich schon wiederholt hinweisen mußte).

Normal Mode

Solange man weder auf <SHIFT 2> (Anführungszeichen) oder <SHIFT INST/DEL> (Insert = Einfügen) drückt, befindet man sich im einfachsten Eingabemodus, dem »Normal Mode« (Normalmodus). Dieser ist durch die Eigenschaft gekennzeichnet, daß alle Steuer-

befehle (also diejenigen Tasten, auf denen nicht sichtbare Zeichen, sondern Steuerfunktionen liegen) unmittelbar ausgeführt werden. <SHIFT CLR/HOME> macht sich also durch sofortiges Löschen des Bildschirms bemerkbar. Gleiches gilt für die Cursorbewegungen oder Änderungen der Schriftfarbe.

Quote Mode

Damit man nun Steuerzeichen in Strings aufnehmen kann, wie schon in der Vorbemerkung erwähnt, gibt es aber noch den »Quote Mode«, oder – deutsch, aber umständlich – »Anführungszeichenmodus«. Diesen kennen Sie natürlich, denn er wird auch im Handbuch beschrieben. Zur Wiederholung: Im Quote Mode werden Steuerzeichen als reverse Buchstaben oder Grafikzeichen dargestellt, nicht jedoch ausgeführt. Eine spätere Ausgabe eines Strings, der solche Steuerzeichen enthält, führt jedoch im Normal Mode zum gewünschten Ergebnis.
In den Quote Mode selbst gelangt man durch einmalige Eingabe eines Anführungszeichens; gibt man ein weiteres ein, befindet man sich wieder im Normal Mode, während erneutes < " > zurück in den Quote Mode führt usw.

Insert Mode

Schließlich gibt es noch diesen Einfügemodus, der in seiner Wirkung dem Quote Mode entspricht – mit der Ausnahme, daß im »Insert Mode« zusätzlich noch <DEL> als reverses Zeichen darstellbar ist. Daraus ergeben sich interessante Tricks, die auch beim Thema »Einzeiler« noch zum Tragen kommen werden. Im allgemeinen ist der Insert Mode jedoch für unsere Zwecke vor allem eine Möglichkeit, Steuerzeichen ohne vorherige Anführungszeichen revers darzustellen.

Wie man die Modi auslöst und beendet

Im kurzen Vergleich rekapitulieren wir noch einmal, auf welche Weise ein bestimmter Eingabemodus betreten wird:

- Normal Mode: Ausgangszustand; kann immer durch <RETURN> oder <SHIFT RETURN> aktiviert werden. Bei Verlassen des Quote Mode oder Insert Mode gelangt man ebenfalls wieder in den Normal Mode zurück. Am einfachsten und bequemsten, wenn auch relativ unbekannt, ist die Möglichkeit, mit <SHIFT RETURN> den Quote Mode zu beenden. Dabei springt der Cursor zwar an den Anfang der nächsten Zeile, aber die Eingabe wird im Gegensatz zu <RETURN> nicht ausgeführt; oft möchte man nämlich bestimmte Cursorbewegungen ausführen, wird aber durch Quote Mode oder Insert Mode davon abgehalten (Cursortasten gelten als Steuertasten und sind somit nur im Normal Mode unmittelbar aktiv). Der Normal Mode bleibt jeweils bis zum nächsten Drücken von < " > oder <INST> bestehen.
- Insert Mode: Entsteht nur aus dem Normal Mode durch Auslösen von <INST>. Werden genauso viele Zeichen eingegeben, wie <INST> gedrückt wurde, ist der Einfügemodus

automatisch zu Ende. Ein Problem für sich stellt die rigorose Reversdarstellung aller Steuerzeichen dar, deren Ausführung gerade beim Editieren des Einfügungstextes hilfreich wäre. Hier sei wiederum auf den SHIFT-RETURN-Trick hingewiesen; dieser führt nach Schaffen des Einfügeplatzes schnell in den Normal Mode, welcher Cursortasten und <DEL> als Editorfunktionen zuläßt.

- Quote Mode: Solange innerhalb einer Zeile (logische Zeile, s. nächster Abschnitt) eine ungerade Zahl (1,3,5...) von Anführungszeichen eingegeben wurde, wobei ein eventuelles Löschen dieser Anführungszeichen durch <DEL> keine Rolle spielt, ist der Quote Mode eingestellt. Bei geraden Anzahlen (0,2,4,6...) hingegen ist der Quote Mode desaktiviert. Der Quote Mode wird aber auch durch <RETURN> bzw. <SHIFT RETURN> verlassen.

1.2.3 Die logischen und echten Zeilen

Dieser kurze Abschnitt soll nur kurz den Begriff »logische Zeile«, der soeben verwendet wurde, beleuchten. Der C64 stellt normalerweise 40 Zeichen pro Bildschirmzeile. Es handelt sich gewissermaßen um eine physikalische Grenze, man spricht hier von einer »echten Bildschirmzeile«. Davon gibt es immer 25 verschiedene.
Für die Programmierpraxis ist jedoch eine andere Einteilung viel wichtiger. So verwaltet der Editor des C64 auch Eingaben von bis zu 80 Zeichen, also zwei echte Bildschirmzeilen. Dies ist eine Notwendigkeit, um längere Basic-Eingaben als 40 Zeichen zu ermöglichen. Die Schwierigkeit liegt nun darin, daß bei Tätigung einer entsprechend langen Eingabe beide Zeilen, die zur Eingabe gehören, intern als »logische Zeile« zusammengefaßt werden. Die praktische Auswirkung können Sie leicht an Ihrer Tastatur nachvollziehen. Drücken Sie in einer Eingabezeile bitte so viele Tasten (also mehr als 40mal), daß die Eingabe auf zwei echte Bildschirmzeilen aufgeteilt wird. Fahren Sie dann mit den Cursortasten auf das erste Zeichen der gesamten Eingabe; wenn Sie nun <SHIFT RETURN> drücken, landet der Cursor nicht – wie man vermuten könnte – in der nächsten echten, sondern der nächsten logischen Bildschirmzeile (also zwei Zeilen unterhalb der vorherigen Position). Wiederholt man das Experiment mit weniger als 40 Zeichen, springt <SHIFT RETURN> nur um eine Zeile tiefer. <SHIFT RETURN> und <RETURN> orientieren sich nämlich im Gegensatz zu den Cursortasten nicht an den echten, sondern an den logischen Bildschirmzeilen.
Die Aufgliederung in echte und logische Bildschirmzeilen läßt sich folgendermaßen definieren: Jede logische Bildschirmzeile besteht mindestens aus einer, höchstens jedoch aus zwei echten Bildschirmzeilen. Eine echte Bildschirmzeile ist durch die Bildschirmbreite, eine logische durch die Maximallänge für Eingaben, nämlich 80, begrenzt – und sogar diese Zahl werden wir im Verlauf des Buches noch überschreiten!

1.2.4 LIST-Schutz leichtgemacht

Eine recht einfache, aber immer wieder faszinierende Methode, ein Programm gegen LIST zu

sichern, soll hier nur kurz angesprochen werden. Geben Sie dazu im leeren Basic-Speicher (NEW) folgende Zeile ein:

```
10 REM
```

Drücken Sie aber noch nicht <RETURN>, sondern lassen Sie den Cursor hinter dem »M« blinken; lösen Sie nun die Kombination <SHIFT> + <L> aus. Am Bildschirm erscheint in Großschrift ein rechtwinkliges Symbol (entfernte Ähnlichkeiten mit dem Buchstaben »L« sind nicht zu leugnen), in der Kleinschrift ein großes L:

```
10 remL
```

Zunächst erscheint dies als nicht gerade sinnvolle, aber auch nicht besonders wirkungsvolle Eingabe. Doch geben Sie einmal LIST ein:

```
list

10 rem
?syntax error
ready.
```

Wir stellen fest: Irgendwie ist der C 64 mit dem geshifteten »L« nicht zufrieden, sondern bricht den LIST-Vorgang genau an dessen Position ab. Kurz gesprochen, es liegt ein Programmierfehler im Basic des C 64 vor, der die Grundlage dieses Tricks bildet; Profis entnehmen alles dem bereits mehrfach erwähnten C 64-Insider-Buch.
Uns soll hier nur die optische Aufbereitung dieser rudimentären LIST-Schutz-Methode interessieren. So ist doch die LIST-Anzeige im derzeitigen Zustand mit dem Nachteil behaftet, daß die Nummer der entsprechenden Zeile (10) sichtbar wird; deren Löschen wiederum würde ja den LIST-Schutz ebenfalls aufheben. Probieren Sie es deshalb mit folgender Eingabe:

```
10 rem"
```

Drücken Sie <SHIFT RETURN>, weil der Quote Mode die nun folgende Operation nicht zuläßt. Fahren Sie mit dem Cursor hinter das Anführungszeichen, lösen Sie siebenmal (!) <SHIFT INST/DEL> aus und geben Sie danach ebensooft <DEL> ein (es erscheinen sieben reverse T's). Anschließend folgt <SHIFT L>; haben Sie sich zwischendrin vertippt, versuchen Sie es nach <SHIFT RETURN> einfach ein weiteres Mal. Geben Sie dann wieder LIST ein, so erscheint folgende veränderte Anzeige, sofern Sie nicht ein modifiziertes C 64-ROM haben (Floppy-Speeder o.ä.):

```
LIST



?SYNTAX ERROR
READY.
```

Jeder Hinweis auf die verantwortliche Zeile bleibt somit aus; die Zeilennummer wird zwar ausgegeben, aber um ein Vielfaches schneller durch DELETE-Codes gelöscht, als es das menschliche Auge wahrnehmen könnte.
Falls Sie vor das »REM SHIFT-L« noch einen sinnvollen Befehl schreiben und entsprechend viele DELETEs einbauen, so perfektionieren Sie damit den Listschutz. Es sei jedoch nicht verschwiegen, daß man auch mit geringen Kenntnissen über den Umgang mit sogenannten Monitorprogrammen derartige Schutzmechanismen nicht nur mühelos durchschaut, sondern auch sekundenschnell aufhebt.

1.2.5 PRINT-Zeilen mit Quote Mode leichter eingeben

Eine konventionelle Lösung, den Bildschirm zu löschen und gleichzeitig eine reverse Kopfzeile auszugeben, besteht etwa in folgender Zeile:

```
10 PRINT CHR$(147);CHR$(18);"COPYRIGHT (C) 1987 BY MARKT & TECHNIK";
      CHR$(146)
```

Dabei werden Steuerzeichen mit folgenden Bedeutungen verwendet:

```
CHR$(147) = CLR
CHR$ (18) = RVS ON
CHR$(146) = RVS OFF
```

Diese Zeile läßt sich nun in mehrfacher Hinsicht optimieren, so daß das Endergebnis sowohl übersichtlicher als auch kürzer und schneller sein wird. Zunächst stellen wir fest, daß CHR$(146) zum Ausschalten des Reversmodus überflüssig ist – schließlich wird nach jedem PRINT, das nicht mit »;« oder »,« schließt, automatisch ein RETURN (CHR$-Code 13) ausgeführt, der ohnehin die Normalschrift einstellt:

```
10 PRINT CHR$(147);CHR$(18);"COPYRIGHT (C) 1987 BY MARKT & TECHNIK"
```

Diese Optimierungsmethode ist zwar simpel, aber in der Praxis denkt man oftmals nicht daran und gibt am Ende einer PRINT-Zeile mehr Steuerzeichen ein als nötig. Nun wollen wir auch den beiden CHR$-Anweisungen zuleibe rücken. Diese lassen sich mit Hilfe des Quote Mode ersetzen, indem Sie folgende Eingabe tätigen (in spitzen Klammern die neuartigen Tastendrücke):

```
10 PRINT "<SHIFT CLR/HOME><CTRL 9>COPYRIGHT (C) 1987
        BY MARKT & TECHNIK"
```

Am Bildschirm sehen Sie <SHIFT CLR/HOME>, den Ersatz für CHR$(147), als reverses Herz oder – in der Kleinschrift – als reverses, großes S. Analog wird <CTRL 9> zu einem reversen R; anstelle von <CTRL 9> können Sie natürlich auch <CTRL R> verwenden, wie schon in 1.1 erwähnt.

Das Ergebnis der neuen Eingabeweise kann sich sehen lassen: Am Bildschirm erscheint eine wesentlich kürzere Zeile, die sowohl im Programmlisting als auch bei der Eingabe deutlich transparenter ist.

1.2.6 Quote Mode bei Strings und DATA-Zeilen

Um auf das Beispiel aus 1.2.5 zurückzukommen, lassen sich alle über PRINT auszugebenden Zeichenketten auch als Strings definieren (solange die Formatierungszeichen »;« und »,« nicht vonnöten sind). Das letzte Beispiel wäre demzufolge auch folgendermaßen denkbar:

```
10 K$="<SHIFT CLR/HOME><CTRL 9>COPYRIGHT (C) 1987 BY MARKT & TECHNIK"
20 PRINT K$
```

Der Vorteil: Die Kopfzeile ist in A$ gespeichert und läßt sich dadurch an mehreren Stellen im Programm kurzerhand durch »PRINT K$« abrufen.
Oftmals werden jedoch Strings, die früher oder später zur Bildschirmausgabe bestimmt sind, nicht mit direkten Zuweisungen (wie Zeile 10 im letzten Programm) belegt, sondern über READ-DATA-Statements.
In der herkömmlichen Syntax bietet DATA keine Möglichkeit, Steuerzeichen dauerhaft zu speichern. Im Insert Mode werden diese zwar dargestellt, aber außerhalb von Anführungszeichen ebenso schnell wieder »vergessen«.
Ein einfacher Ausweg hilft über dieses Dilemma hinweg: Gibt man in DATA-Zeilen Anführungszeichen an, so begrenzen diese einen String; dies ist schon allein deshalb notwendig, weil sonst ein Komma innerhalb einer Zeichenkette nicht von einer DATA-Abgrenzungskennung zu unterscheiden wäre. Folgende Eingabe zeigt den Einsatz des Quote Mode für DATA-Zeilen:

```
10 READ K$
20 PRINT K$
30 DATA "<SHIFT CLR/HOME><CTRL 9>COPYRIGHT (C) 1987 BY MARKT & TECHNIK"
```

Als sogenannte »Freeware« (frei kopierbare Software, anderer Name für »Public Domain«) verbreitet Commodore eine sehr populäre C64-Version des Gesellschaftsspiels »Monopoly«, die – nur eine persönliche Empfehlung – mir selber schon seit mehreren Jahren ausgezeichnet gefällt!
Wenn Sie dieses lehrreiche Programmierbeispiel besitzen und listen lassen, sehen Sie dort, wie intensiv die Steuerzeichen in DATA-Zeilen verwendet wurden: Jeder Straßenname bekommt seine eigene Zeichenfarbe als Steuerzeichen am Stringanfang zugewiesen. Ganz klar stellt dies die einfachste und effektivste Verfahrensweise dar!

1.2.7 Steuerzeichen mit Supertrick

Im folgenden lernen Sie einen Eingabetrick kennen, der die Eingabe aller (!) vorhandenen Steuerzeichen zuläßt. Dabei geht es im wesentlichen darum, die Steuerzeichen bereits bei der Eingabe als reverse Zeichen einzugeben.
Bei der Ausgabe von Steuerzeichen im Quote Mode über den Befehl

```
PRINT CHR$(34);CHR$(steuercode)
```

erscheinen fast alle Zeichen als reverse Symbole. Kennt man diese, so ist es auch keine Schwierigkeit mehr, sie bei der Eingabe einzusetzen. Als Beispiel wollen wir den Code CHR$(142), der auf den Groß-/Grafik-Schriftmodus schaltet, durch ein Steuerzeichen ersetzen.
Zunächst interessiert uns das Aussehen von CHR$(142) im Quote Mode:

```
PRINT CHR$(34);CHR$(142)
```

liefert ein Anführungszeichen (Code 34) sowie ein reverses <SHIFT N>. Geben wir nun folgende Zeile ein und drücken erst einmal <RETURN>:

```
10 PRINT "
```

Anschließend bewegen wir den Cursor wieder so weit zurück, bis er rechts vom Anführungszeichen steht; aufgrund des vorausgegangenen <RETURN> sind wir allerdings dem Quote Mode entkommen. Deshalb schalten wir mit <CTRL 9> oder <CTRL R> die Reversschrift ein und drücken <SHIFT N>. Anschließend lösen wir erneut <RETURN> aus.
Die Ausführung dieses Befehls entspricht nun PRINT CHR$(142), doch die Eingabe war um einiges raffinierter.
Nach dem beschriebenen Verfahren läßt sich prinzipiell jedes Steuerzeichen erreichen. Selbst <SHIFT RETURN> als reverses, geshiftetes M ist möglich; bei späteren LIST-Versuchen wird es jedoch ausgeführt und verunstaltet den Bildschirmaufbau, da alle Steuerzeichen nach <SHIFT RETURN> im Normalmodus zur Bildschirmausgabe kommen und dadurch ebenfalls ihre Wirkung zeigen – was wiederum als LIST-Schutz praktisch ist.
Noch wichtiger für die Praxis ist jedoch die Möglichkeit, auch Drucker-Steuerzeichen aller Art über den Quote Mode einzugeben. Langwierige Steuersequenzen in Basic-Zeilen nach dem Schema

```
PRINT #4,CHR$(27)CHR$(126)CHR$(49)CHR$(27)CHR$(82)CHR$(0) ....
```

werden dadurch um einen Faktor 8–9 kürzer.

1.2.8 Filenamen mit Steuerzeichen

Grundsätzlich eignet sich für die Steuerzeichen und die beschriebenen Trick-Eingabemethoden jedes Betätigungsfeld, das mit Zeichenketten zusammenhängt. Eine besonders interessante Anwendung stellen die Dateinamen für Kassette und Diskette dar.

a. Kassette

Jeder String von 16 Zeichen Länge kann beim Abspeichern auf Diskette eingesetzt werden; für die Steuerzeichen besteht keinerlei Limit. So ist es beispielsweise denkbar, <SHIFT CLR/HOME> an den Anfang eines Filenamens zu setzen. Der Effekt liegt dann darin, daß bei der FOUND-Anzeige zunächst der Bildschirm gelöscht und dann links oben der eigentliche Programmname angezeigt wird. Der Einbau von Farbsteuerzeichen, <RVS ON> usw. bietet sich gleichermaßen an.

Hier ein Eingabebeispiel:

```
SAVE"<SHIFT CLR/HOME><CTRL H><CTRL N><RVS ON>c16<RVS OFF>merge"
```

Beim Laden wird dann der Bildschirm gelöscht, die Tastenkombination <SHIFT C=> verboten, auf Kleinschrift geschaltet sowie der Text »c16« revers und »merge« normal dargestellt.

b. Diskette

Das beschriebene Verfahren ist in gleicher Weise auch auf Diskette anwendbar, doch sind dabei zwei entscheidende Punkte zu berücksichtigen:

1. Während bei Kassettenfiles die Angabe des Filenamens nicht nötig ist, muß man bei Disketten-Dateien zumindest über so viele Zeichen verfügen, daß die Datei eindeutig zu bestimmen ist. Also denken Sie auch immer an den Anwender, der das Programm schließlich laden möchte! Programminterne Unterroutinen hingegen darf man ohne weiteres auf diese Weise schützen.
2. Die FOUND-Anzeige von Kassette erleichtert die Steuerzeichen-Anwendung vor allem dadurch, daß der Filename ohne Anführungszeichen (im Normal Mode) ausgegeben wird. Das Directory einer Diskette hingegen zeigt die Filenamen innerhalb von Anführungszeichen an.

Ein Beispiel für Steuerzeichen in Filenamen ist das Zeichenprogramm Koala-Painter, dessen Grafikfiles mit einem Farbsteuerzeichen (als Erkennungsmarke für den Koala-Painter) beginnen. Beim Kopieren mit herkömmlichen Programmen ergibt sich dadurch ein Farbwechsel, wenn die Filenamen – im Gegensatz zum Directory – außerhalb von Anführungszeichen stehen.

1.2.9 INPUT und Steuerzeichen

Im allgemeinen entsprechen INPUT-Eingaben für laufende Basic-Programme dem Programm-Eingabemodus. Dies hat den Vorteil, daß man Steuerzeichen wie <CLR> oder die Farbwechsel ausführen kann, um damit auf die Bildschirmgestaltung des aktuellen Programms auch als Anwender gewissen Einfluß zu nehmen. Zum anderen gibt es Programme, die die <SHIFT C=>-Umschaltung durch CHR$(8) = <CTRL H> blockieren; hierfür genügt in einer

INPUT-Eingabe ein kurzes <CTRL I>, und schon ist <SHIFT C=> vorerst wieder freigegeben. Dies ist aber nur in reinen INPUT-Eingaben möglich, nicht hingegen in speziellen, vom Programm vorgesehenen Eingaberoutinen, die letztlich auf GET basieren.
Für eigene Programme, die einen professionellen Touch haben sollen, empfehle ich Ihnen deshalb einen sogenannten Initialisierungsstring zu definieren; dieser sollte alle Steuerzeichen zur Herstellung des gewünschten Ausgabezustandes (Schriftmodus, <SHIFT C=>-Sperre, Zeichenfarbe) enthalten und nach allen »kritischen« Aktionen (wie INPUT-Eingaben) ausgeführt werden. Damit wird der Schaden, den gezielte wie auch unabsichtliche INPUT-Eingaben von Steuerzeichen anrichten, auf ein absolutes Minimum reduziert. Noch besser ist natürlich die Verwendung einer speziellen Eingaberoutine, wie Sie sie in Kapitel 3.1.2 erhalten.

1.3 Tastaturprogrammierung für Insider

Waren die bisherigen Ausführungen in diesem Kapitel vor allem an Einsteiger gerichtet, geht es in diesem Abschnitt um so stärker ans Eingemachte. Die Maschinenprogrammierer erhalten ablauffertige Beispiele für die Einbindung eigener Steuerzeichen, worauf in bisheriger Literatur noch nicht eingegangen wurde, sowie für die Funktionstastenbelegung und eine Routine, die die Cursorbewegungen in Abhängigkeit vom Joystick durchführt.

1.3.1 Der Joy-Cursor

Gerade als dieser Abschnitt entsteht, lese ich in einer Computerzeitschrift vom sogenannten »Icontroller«; dabei handelt es sich um einen kleinen Joystick, der rechts an die Tastatur angebracht wird und nicht gerade für Ruck-, Reiß- und Zerr-Telespiele, sondern vielmehr für die GEOS-Bedienung entworfen wurde. Hier ist nun ein Utility, jeden Joystick (auch den Icontroller) zur bequemen Cursorsteuerung einzusetzen, wie es beim sogenannten »Enterprise«-Computer standardmäßig vorgesehen ist. Das Utility wird über

`LOAD "JOYCURSOR $C000",8,1` geladen und nach

`NEW` über

`SYS 49152`

aktiviert und, solange es sich im Speicher befindet, auch neu gestartet. Fortan bewegt auch der Joystick in Port 2 den Cursor, bis das Programm beispielsweise über <RUN/STOP RESTORE> desaktiviert wird. Da die Cursorbewegungen nicht anders wirken, als wenn die entsprechenden Cursortasten gedrückt würden, erscheinen im Quote Mode die jeweiligen Steuerzeichen.
Nun zur Funktionsweise des kleinen, aber wirkungsvollen und lehrreichen Maschinenprogramms. Ganz grob gesagt, bindet es sich in den Systeminterrupt ein und prüft dadurch regel-

mäßig, ob eine Joystickbewegung festzustellen ist. In diesem Fall wird sie in das entsprechende Steuerzeichen »übersetzt« und dieses schließlich in den Tastaturpuffer (Näheres darüber in 1.4) geschrieben.

Nun zum Quelltext im Format des Assemblers Hypra-Ass, welcher sich auch auf der Rückseite der Masterdiskette befindet.

Listing 1.1: Quelltext zum JOYCURSOR-Utility

```
100   -;
110   -; ******************************
120   -; *                            *
130   -; *     CURSORSTEUERUNG MIT    *
140   -; *                            *
150   -; *       J O Y S T I C K      *
160   -; *                            *
170   -; *    (PORT 2) IM INTERRUPT   *
180   -; *                            *
190   -; ******************************
200   -;
210   -;
220   -             .BA $C000        ; START: SYS 49152
230   -;
240   -.GL  PORT2 = 56320            ; CIA-REGISTER
250   -.GL  NDX   = 198              ; ANZAHL DER ZEICHEN IM TASTATURPUFFER
260   -.GL  CINV  = $0314            ; IRQ-VEKTOR DES BETRIEBSSYSTEMS
270   -;
280   -.MA ALTER_IRQ
290   -             JMP $EA31
300   -.RT
310   -;
320   -;
330   -; *** INITIALISIERUNG ***
340   -;
350   -;
360   -             SEI              ; IRQ VERHINDERN
370   -             LDA #<(NEWIRQ)   ; NEUE
380   -             LDY #>(NEWIRQ)   ; IRQ-
390   -             STA CINV         ; ROUTINE
400   -             STY CINV+1       ; AKTIVIEREN
410   -             LDA #6           ; ZAEHLER
420   -             STA ZAEHLER      ; INITIALISIEREN
430   -;
440   -             CLI              ; IRQ ZULASSEN
450   -             RTS              ; RUECKSPRUNG
460   -;
470   -;
480   -; *** ZAEHLVARIABLE (COUNTER) ***
490   -;
500   -ZAEHLER      .BY 6
510   -;
520   -;
530   -; *** NEUE IRQ-ROUTINE ***
540   -;
550   -NEWIRQ       DEC ZAEHLER      ; WARTEZEIT ABGELAUFEN (NUR ALLE 6 INTERRUPTS) ?
560   -             BEQ WEITER       ; JA (Z=1): NOCH NICHT ABBRECHEN
570   -             ... ALTER_IRQ    ; ALTER IRQ
580   -; ---------------------
590   -;
600   -WEITER       LDA #6           ; ZAEHLER WIEDER
610   -             STA ZAEHLER      ; INITIALISIEREN
620   -;
630   -             LDX NDX          ; TASTATURPUFFER
640   -             CPX #$08         ; VOLL ?
```

```
650  -            BCC MOVE        ; NEIN (C=0): BEWEGUNG AUSFUEHREN
660  -;
670  -            ... ALTER_IRQ   ; ALTER IRQ
680  -; ------------------------
690  -;
700  -MOVE        LDA PORT2       ; JOYSTICK-ZUSTAND AUSLESEN
710  -            ROR             ; URSPRUENGLICHES BIT 0 TESTEN
720  -            BCC UNTEN       ; B0=0 (C=0): JOYSTICK NACH UNTEN
730  -            ROR             ; URSPRUENGLICHES BIT 1 TESTEN
740  -            BCC OBEN        ; B1=0 (C=0): JOYSTICK NACH OBEN
750  -            ROR             ; URSPRUENGLICHES BIT 2 TESTEN
760  -            BCC LINKS       ; B2=0 (C=0): JOYSTICK NACH LINKS
770  -            ROR             ; URSPRUENGLICHES BIT 3 TESTEN
780  -            BCC RECHTS      ; B3=0 (C=0): JOYSTICK NACH RECHTS
790  -            ... ALTER_IRQ   ; ALTER IRQ
800  -; ------------------------
810  -;
820  -OBEN        LDA #"(DOWN)"   ; <CRSR DOWN>
830  -            BNE AUSGABE     ; AUSGEBEN
840  -; ------------------------
850  -;
860  -UNTEN       LDA #"(UP)"     ; <CRSR UP>
870  -            BNE AUSGABE     ; AUSGEBEN
880  -; ------------------------
890  -;
900  -LINKS       LDA #"(LEFT)"   ; <CRSR LEFT>
910  -            BNE AUSGABE     ; AUSGEBEN
920  -; ------------------------
930  -;
940  -RECHTS      LDA #"(RGHT)"   ; <CRSR RIGHT> AUSGEBEN
950  -; ------------------------
960  -;
970  -; ***  Z E I C H E N   I N  ***
980  -; ***     TASTATURPUFFER     ***
990  -; ***   S C H R E I B E N    ***
1000 -;
1010 -AUSGABE     LDX NDX         ; OFFSET HOLEN
1020 -            STA $0277,X     ; BYTE SCHREIBEN
1030 -            INC NDX         ; ZAEHLER ERHOEHEN
1040 -            ... ALTER_IRQ   ; ALTER IRQ
1050 -; ------------------------
```

Die Zeilen 240–260 definieren die Konstanten PORT2, NDX und CINV. PORT2 ist die Adresse des CIA-Registers, das über den Joystickzustand in Port 2 berichtet; ist ein Bit gelöscht (!), so fordert dies eine Bewegung in die entsprechende Richtung. NDX gibt die Anzahl derjenigen Zeichen an, die bereits im Tastaturpuffer stehen und weist damit gleichzeitig als Index vom Beginn des Tastaturpuffers auf die Position für den nächsten (kommenden) Eintrag. CINV ist der Interrupt-Vektor des Betriebssystems, oft auch als IRQVEC bezeichnet; normalerweise zeigt er nach $ea31, wohin wiederum das Makro »AlterIRQ« verzweigt.

Die Initialisierung (360–450) richtet CINV auf die neue IRQ-Routine bei NEWIRQ (Zeile 550), initialisiert den ZAEHLER (enthält die Anzahl der IRQ-Aufrufe, bis wieder der Joystick abgefragt wird) und springt schließlich an die aufrufende Routine (bzw. ins Basic) zurück.

Der Zähler steht zwischen Initialisierung und der eigentlichen IRQ-Routine, welche diesen jeweils herunterzählt. So wird alle 6 IRQ-Aufrufe, also etwa 10mal pro Sekunde, der Programmteil ab WEITER (Zeile 600) durchlaufen. Die eigentliche Joystickabfrage vollzieht sich

also erst dort. Einzige Voraussetzung für die Umwandlung der Joystick-Bewegung in ein Steuerzeichen ist noch, daß der Tastaturpuffer Platz für einen weiteren Eintrag hat (Zeilen 630–670). Dann schließt sich eine überaus geschickte, unübertroffene Abfolge von Bit-Tests an; der Registerinhalt wird jeweils nach rechts verschoben, um jeweils das unterste Bit zu testen und dadurch sukzessive alle Bits zu überprüfen. Ist eine bestimmte Joystick-Bewegung erkannt, schreibt die entsprechende Sonderbehandlung (UNTEN, OBEN, LINKS, RECHTS) das richtige Steuerzeichen ab Zeile 1000 (AUSGABE) in den Tastaturpuffer, um schließlich ebenfalls in den alten IRQ zurückzukehren. Interessant sind die BNE-Befehle in den Zeilen 830, 870 und 910; diese dienen zum Überspringen aller darauffolgenden Anweisungen; da in den Befehlen davor generell Werte < > 0 geladen werden, verzweigen die BNE-Befehle unter Garantie. Die BNEs sind also Pseudo-JMPs. Eine andere Möglichkeit bestünde darin, BNE AUSGABE global durch BIT $2C (Opcode für BIT $xxxx) zu ersetzen; dann würden ebenfalls die einzelnen LDAs übergangen, da BIT den Akku nicht verändert. Die BIT-Adressen wären in diesem Fall nur die versteckten LDAs.

Der bessere Programmierstil ist jedoch die Verwendung von BNE, wie es in Listing 1.1 zu sehen ist.

1.3.2 Benutzerdefinierte Steuerzeichen

Die Definition eigener Zeichensätze ist für viele Programmierer gang und gäbe. Als mindestens so hilfreich, aber noch sehr unbekannt erweist sich die Eigenentwicklung von Steuerzeichen, die dann bei BSOUT ($FFD2) Berücksichtigung finden. Dabei gibt es ausreichend viele, noch unbelegte ASCII-Codes, die förmlich darauf warten, eine sinnvolle Funktion repräsentieren zu dürfen. Der programmtechnische Aufwand ist nicht sehr groß und sogar zu vernachlässigen, wenn Sie das im folgenden vorgestellte Grundgerüst verwenden.

Der Vorteil liegt besonders darin, daß einmal programmierte Steuerfunktionen fortan modular zur Verfügung stehen. Theoretisch könnte man sich seine eigene Steuerzeichen-Bibliothek anlegen.

Laden Sie zur Demonstration ein kleines Maschinenprogramm:

```
LOAD "NEUE STEUERZEICH",8,1
NEW
SYS 49152
```

Von nun an ist das Betriebssystem um die folgenden Steuercodes reicher:

```
CHR$(2) = Invertieren des Bildschirms
CHR$(4) = Löschen der aktuellen Cursorzeile
CHR$(7) = Klingelton
CHR$(27)= Quote Mode verlassen (Go to Normal Mode)
```

Im Quote Mode werden die neuen Steuerzeichen folgendermaßen eingegeben:

```
CHR$(2) = <CTRL B>
CHR$(4) = <CTRL D>
CHR$(7) = <CTRL G>
CHR$(27)= <CTRL >     ******* <CTRL eckige Klammer zu>
```

Möglich ist jedoch nur die Eingabe der Steuerzeichen, nicht ihre unmittelbare Ausführung. Warum die neuen Steuerzeichen auch im Normal Mode ohne Wirkung bleiben, werden wir noch ergründen. Zur Demonstration sollten Sie jedoch über

```
PRINT CHR$(2);CHR$(7)
```

beispielshalber zwei Codes ausprobieren: Der Bildschirm invertiert sich, und dies wird mit einem Klingelzeichen akustisch untermalt.
Die beiden anderen Steuerzeichen sind am besten in Beispielen einzusetzen:

```
100 PRINT "DIES IST DIE AKTUELLE ZEILE";CHR$(4)
```

(Ein Text wird ausgegeben und sogleich gelöscht).

```
110 PRINT CHR$(34);CHR$(147)
```

gibt ein Anführungszeichen (Quote Mode!) und ein reverses Herz aus, wogegen die Einfügung von CHR$(27) den Quote Mode beendet und somit das Bildschirmlöschen zuläßt:

```
110 PRINT CHR$(34);CHR$(27);CHR$(147)
```

Die programmtechnischen Grundlagen

Zunächst soll uns das Funktionsprinzip klar werden:
Damit die neuen Steuerzeichen erkannt werden, schalten wir uns durch Umbiegen des Vektors IBSOUT (für die Kernel-Routine BSOUT) »vor«; findet unser Programm ein eigenes Steuerzeichen, so ruft es die dazugehörige Routine auf, danach übernimmt – was bei allen anderen Zeichencodes sofort geschieht – die alte BSOUT-Behandlung den Rest.
Anhand des Quelltextes läßt sich dies am besten nachvollziehen.
Die Zeilen 500–570 enthalten die Festlegung der Konstanten. IBSOUT ist der Vektor für die Kernel-Routine BSOUT ($ffd2) und weist normalerweise nach OLDBSOUT ($f1ca). DELLIN ($e9ff) ist ein nützlicher ROM-Einsprung, welcher die Bildschirmzeile, deren Nummer (0–24) im X-Register steht, mit Leerzeichen überschreibt, also im Klartext: diese Zeile löscht. QTSW ist ein Betriebssystem-Flag, das mit Werten < > 0 die Aktivität des Quote Mode oder Insert Mode signalisiert; 0 steht für den Normal Mode. TBLX enthält jederzeit die Nummer derjenigen Zeile, in welcher gerade der Cursor steht. Last not least: ANZAHL enthält die Menge der neu eingebundenen Steuerzeichen und erhöht in starkem Maße die Flexibilität von NEUE STEUERZEICHEN; dieses Programm sollten Sie ja, wie schon gesagt, auch als Basis für eigene Entwicklungen verwenden können.

Listing 1.2: Quelltextbeispiel »NEUE STEUERZEICHEN«

```
100  -;
110  -; ********************************
120  -; *                              *
130  -; *      NEUE STEUERZEICHEN      *
140  -; *      ==================      *
150  -; *                              *
160  -; * EIN: SYS 49152               *
170  -; * AUS: <RUN/STOP>+<RESTORE>    *
180  -; *                              *
190  -; ********************************
200  -; *                              *
210  -; * CHR$(2) = <CTRL>+<B> :       *
220  -; *   BILDSCHIRM INVERTIEREN     *
230  -; *                              *
240  -; * CHR$(4) = <CTRL>+<D> :       *
250  -; *   AKTUELLE ZEILE LOESCHEN    *
260  -; *                              *
270  -; * CHR$(7) = <CTRL>+<G> :       *
280  -; *   KLINGELTON ERZEUGEN        *
290  -; *                              *
300  -; * CHR$(27)= <CTRL>+<]> :       *
310  -; *   QUOTE MODE VERLASSEN       *
320  -; *                              *
330  -; ********************************
340  -; *                              *
350  -; * PRINZIP: VERBIEGEN DES       *
360  -; *          IBSOUT-VEKTORS      *
370  -; *                              *
380  -; *                              *
390  -; * (W) 25.09.1987 BY            *
400  -; *     FLORIAN MUELLER          *
410  -; *                              *
420  -; * ASSEMBLER: HYPRA-ASS         *
430  -; *            (C64)             *
440  -; *                              *
450  -; ********************************
460  -;
470  -;
480  -;
490  -.BA $C000
500  -;
510  -;
520  -.GL IBSOUT    = $0326     ; BSOUT-VEKTOR
530  -.GL OLDBSOUT  = $F1CA     ; NORMALER IBSOUT-INHALT
540  -.GL DELLIN    = $E9FF     ; ZEILE (X) LOESCHEN
550  -.GL QTSW      =   $D4     ; QUOTE-MODE-FLAG
560  -.GL TBLX      =   $D6     ; AKTUELLE CURSORZEILE
570  -.GL ANZAHL    = 4         ; ANZAHL DER ZUSAETZLICHEN STEUERZEICHEN
580  -;
590  -;
600  -;
610  -; *** INITIALISIERUNG ***
620  -;
630  -;
640  -          LDA #<(NEWBSOUT); VEKTOR
650  -          LDY #>(NEWBSOUT); IBSOUT
660  -          STA IBSOUT      ; AUF NEWBSOUT
670  -          STY IBSOUT+1    ; UMLENKEN
680  -          RTS             ; UND ZURUECKSPRINGEN
690  -;
700  -;
710  -;
720  -; *** RETTUNGSSPEICHER FUER ***
730  -; ***    PROZESSOR-REGISTER  ***
```

```
740  -;
750  -;
760  -XGET       .BA XGET+1      ; RETTUNGSSPEICHER FUER X-REGISTER
770  -AKKU       .BA AKKU+1      ; RETTUNGSSPEICHER FUER AKKUMULATOR
780  -YGET       .BA YGET+1      ; RETTUNGSSPEICHER FUER Y-REGISTER
790  -;
800  -;
810  -;
820  -; *** MAKRO FUER RUECKSPRUNG ***
830  -; ***  AUS EIGENEM NEWBSOUT  ***
840  -;
850  -.MA RETURN (AKKU,XGET,YGET,OLBDSOUT)
860  -           LDA AKKU        ; PROZESSOR-
870  -           LDX XGET        ; REGISTER
880  -           LDY YGET        ; WIEDERHERSTELLEN
890  -           JMP OLDBSOUT    ; WEITER WIE ALTES BSOUT
900  -.RT
910  -;
1000 -;
1010 -;
1020 -; *** NEUE BSOUT-BEHANDLUNG ***
1030 -;
1040 -;
1050 -;
1060 -NEWBSOUT   STX XGET        ; X-REGISTER RETTEN
1070 -           LDX #ANZAHL-1   ; SUCHINDEX INITIALISIEREN
1080 -VGL        CMP CHARS,X     ; VERGLEICH MIT TABELLE
1090 -           BEQ FOUND       ; UEBEREINSTIMMUNG (Z=1): ZEICHEN AUSFUEHREN
1100 -           DEX             ; INDEX WEITERZAEHLEN
1110 -           BPL VGL         ; GEGEBENENFALLS WEITERE SUCHE IN DER TABELLE
1120 -           LDX XGET        ; X-REGISTER WIEDERHERSTELLEN
1130 -           JMP OLDBSOUT    ; UND ZEICHEN NORMAL AUSGEBEN
1500 -;
1510 -; ===============================
1520 -; BEHANDLUNG EINES NEUEN ZEICHENS
1530 -; ===============================
1540 -;
1550 -FOUND      STA AKKU        ; AKKUMULATOR RETTEN
1560 -           STY YGET        ; Y-REGISTER RETTEN
1570 -           LDA LOW,X       ; SPRUNGBEFEHL
1580 -           STA SPRUNG+1    ; AUF ENTSPRECHENDE
1590 -           LDA HIGH,X      ; ROUTINE
1600 -           STA SPRUNG+2    ; LENKEN
1610 -SPRUNG     JMP $0000       ; SELBSTMODIFIKATION!
1620 -;
2000 -;
2010 -; *** TABELLEN ***
2020 -;
2030 -LOW        .BY <(D),<(B),<(ESC),<(G)
2040 -HIGH       .BY >(D),>(B),>(ESC),>(G)
2050 -CHARS      .BY $04,$02,$1B,$07; D,B,ESC,G
2060 -;
2070 -;
5000 -;
5010 -;
5020 -; *** CHR$(4) ***
5030 -;
5040 -;
5050 -D          .BA D           ; AKTUELLE ZEILE LOESCHEN
5060 -           LDX TBLX        ; NUMMER DER AKTUELLEN ZEILE HOLEN
5070 -           JSR DELLIN      ; UND ZEILE LOESCHEN
5080 -           ... RETURN(AKKU,XGET,YGET,OLDBSOUT)
5090 -;
6000 -;
6010 -;
6020 -; *** CHR$(2) ***
6030 -;
6040 -;
```

```
6050 -B          .BA B          ; BILDSCHIRM INVERTIEREN
6060 -           LDA #<(1024)   ; LOW-BYTES AUF
6070 -           STA READ+1     ; BILDSCHIRMSPEICHERANFANG
6080 -           STA WRITE+1    ; SETZEN
6090 -           LDA #>(1024)   ; HIGH-BYTES AUF
6100 -           STA READ+2     ; BILDSCHIRMSPEICHERANFANG
6110 -           STA WRITE+2    ; SETZEN
6120 -;
6130 -           LDY #0         ; OFFSET INITIALISIEREN
6140 -READ       LDA $FFFF,Y    ; BILDSCHIRMCODE LADEN
6150 -           EOR #$80       ; REVERS-BIT >>FLIPPEN<<
6160 -WRITE      STA $FFFF,Y    ; UND ZURUECKSCHREIBEN
6170 -           INY            ; OFFSET ERHOEHEN
6180 -           BNE READ       ; SCHLEIFE FORTSETZEN
6190 -           INC READ+2     ; HIGH-BYTES
6200 -           INC WRITE+2    ; ERHOEHEN
6210 -           LDA WRITE+2    ; UND ZWECKS PRUEFUNG LESEN
6220 -           CMP #>(2048)   ; FERTIG ?
6230 -           BCC READ       ; NEIN (C=0): WEITER IN SCHLEIFE
6240 -;
6250 -           ... RETURN(AKKU,XGET,YGET,OLDBSOUT)
6260 -;
6270 -;
7000 -;
7010 -;
7020 -; *** CHR$(27) ***
7030 -;
7040 -;
7050 -ESC        .BA ESC        ; ESCAPE
7060 -           LDA #0         ; LOESCHWERT IN
7070 -           STA QTSW       ; QUOTE-MODE-FLAG SCHREIBEN
7080 -           STA AKKU       ; CHR$(0) ZURUECKGEBEN
7090 -           ... RETURN(AKKU,XGET,YGET,OLDBSOUT)
7100 -;
8000 -;
8010 -;
8020 -; *** CHR$(7) ***
8030 -;
8040 -;
8050 -G          .BA G          ; BELL (KLINGELZEICHEN)
8060 -           LDA #$0F       ; SIGNALTON-
8070 -           STA $D418      ; MODUL
8080 -           LDA #0         ; WURDE AUS
8090 -           STA $D405      ; DEM PROGRAMM
8100 -           LDA #$F7       ; >>MSE<<
8110 -           STA $D406      ; (64'ER-EINGABEHILFE)
8120 -           LDA #$11       ; >>ENTLIEHEN<<
8130 -           STA $D404
8140 -           LDA #$32
8150 -           STA $D401
8160 -           LDA #0
8170 -           STA $D400
8180 -           LDY #$80       ; WARTE-KONSTANTE
8190 -SOUND1     LDX #$FF       ; WARTESCHLEIFE
8200 -SOUND2     DEX            ; LOW-BYTE ABGELAUFEN ?
8210 -           BNE SOUND2     ; NEIN (Z=0): INNERE WARTESCHLEIFE FORTSETZEN
8220 -           DEY            ; HIGH-BYTE ABGELAUFEN ?
8230 -           BNE SOUND1     ; NEIN (Z=0): AEUSSERE WARTESCHLEIFE FORTSETZEN
8240 -;
8250 -           LDA #$10       ; TON WIEDER
8260 -           STA $D404      ; ABSTELLEN
8270 -;
8280 -           ... RETURN(AKKU,XGET,YGET,OLDBSOUT)
8290 -;
8300 -;
```

Den Vektor IBSOUT verbiegen die Zeilen 600–680 nach NEWBSOUT (ab Zeile 1000) und springen an die Aufrufstelle (bei SYS-Einsprung ist dies der Basic-Interpreter) zurück. Dahinter liegen die Speicherplätze, in welchen die Prozessor-Register gerettet werden. Das Makro RETURN (Zeilen 800–910) gibt diese zurück und springt die alte BSOUT-Routine (OLDBSOUT $f1ca) an. RETURN schließt jede Sonderbehandlung für ein selbstdefiniertes Steuerzeichen ab: Die Prozessorregister bei Eintritt in die »umgebogene« BSOUT-Routine werden restlos wiederhergestellt, das Original-BSOUT bekommt das Wort erteilt.
Im neuen BSOUT-Vorspann ermitteln die Zeilen 1000–1130, ob der auszugebende Code (im Akku enthalten) in der Tabelle der neuen Sonderzeichen steht. Falls ja, so wird die Position als Index für die Adresse in 1500–1620 zum Ansprung der dazugehörigen Steuerzeichen-Routine eingesetzt. Die verwendete Technik der Selbstmodifikation (der Befehl JMP $0000 in Zeile 1610 wird in 1580 und 1590 verändert) erleichtert die Programmierung erheblich und steht in Kapitel 10 ausführlich beschrieben. Die verwendeten Tabellen (Steuerzeichen-Codes, Low- und High-Bytes der Adressen) liegen bei den Zeilen 2000–2070 unmittelbar hinter dem selbstmodifizierenden Teil.
Die einzelnen Routinen schließlich sind von geringerer Bedeutung: D (5000–5090) holt die Nummer der aktuellen Cursorzeile und löscht diese; B (6000–6270) bedient sich in einer Schleife, die den Bildschirmspeicher durchgehend invertiert (Umblendung von Bit 7), wiederum der Selbstmodifikation; ESC (7000–7100) löscht das Quote-Mode-Flag durch Setzen auf Null; G (8000–8300) erzeugt den Klingelton.
Mit Sicherheit ließen sich noch Dutzende weiterer Steuerzeichen realisieren; auch die Abwandlung bereits bestehender ASCII-Codes zugunsten eigener Modifikationen ist ein reizvolles Thema. An Kenntnissen benötigen Sie außer der hier beschriebenen Einbindungstechnik lediglich ein ROM-Listing; ich möchte Ihnen hierfür mein Buch »C 64 für Insider« ans Herz legen, welches wirklich ausführlich jede einzelne Steuerzeichen-Routine zerlegt – sowohl im ROM-Listing selbst als auch in einem ausführlichen Dokumentations-Fließtext. Eine ausführliche Speicherbelegung ist inbegriffen und erschließt weitere Anwendungsgebiete.

1.3.3 Funktionstastenbelegung

Folgende Möglichkeiten bestehen zur Funktionstastenbelegung:

- Die IRQ-Routine läßt sich dahingehend modifizieren, daß sie eine »Aktionstaste« wie <F1> sofort erkennt und automatisch eine dazugehörige Routine startet. Dieses Verfahren ist stark mit dem Programm aus 1.3.1 verwandt.
- Der IKEYLOG-Vektor weist auf die Routine zur Tastaturdekodierung und ist am ergiebigsten für Änderungen dieser Art. Näheres in »C 64 für Insider« sowie im Programm, das in 1.5 vorgestellt wird.

1.4 Direkt oder indirekt?

Die in diesem Abschnitt beschriebenen Programmiertechniken sind vor allem für Basic-Programmierer von unschätzbarem Wert; wer den sogenannten Tastaturpuffer effizient zu gebrauchen weiß, löst viele Probleme des Programmieralltags mit wenigen, aber unorthodoxen Handgriffen. Dabei geht es um nichts weiter, als dem Betriebssystem die Auslösung bestimmter Tastendrücke vorzugaukeln...

1.4.1 Big Brother Betriebssystem is watching you!

Als Einsteiger hat man beim Ablauf vieler Programme oft das Gefühl, daß der Computer voll mit den Berechnungen des Programms beschäftigt ist und stur vor sich hinrechnet, aber darüber hinaus nichts wahrnimmt. Weit gefehlt, denn das Betriebssystem hat jede 1/60 Sekunde ein wachsames Auge auf alle Tastendrücke auf der Computertastatur oder der Datasette. Ohne daß man es merkt, wird jeder Tastendruck auch außerhalb des Eingabemodus sehr wohl notiert und später berücksichtigt. Geben Sie doch einmal zur Beweisführung folgenden Befehl ein:

```
FOR F=1 TO 1000:NEXT
```

Durch diese Anweisung wird der C64 für eine gewisse Zeit lahmgelegt. Doch drücken Sie dabei ein paar Tasten, aber nicht allzu viele (maximal werden zehn Tastendrücke gespeichert). Nach Ablauf der Schleife (bitte nicht mit <RUN/STOP> unterbrechen) erscheinen dann Ihre zuvor getätigten Eingaben am Bildschirm. Dieses Phänomen ist auf die »dynamische Tastenabfrage« zurückzuführen; die Tastatur wird nämlich nicht nur bei Bedarf, also während INPUT, GET oder sonstigen Eingaben (der Direkteingabemodus des Basic-Interpreters sei hier einfach mit INPUT gleichgesetzt), sondern permanent überprüft. Eine Eigenschaft, die ich bei manch anderem Computer schmerzlich vermisse!

1.4.2 Bitte nicht drängeln – die Tastatur-Warteschlange

Am Beispiel des Basic-Befehls GET kennen Sie bestimmt die Abfrage der Tastatur: Man erhält jeweils für eine gedrückte Taste deren ASCII-Code (bei GET einen String des betreffenden Inhalts) zurück. Es würde in diesem Zusammenhang zu weit führen, alle (hochinteressanten) Schritte zu erläutern, die von der mechanischen Einordnung eines Tastendrucks bis zum fertigen String verantwortlich sind (siehe »C64 für Insider«).

Für Basic-Zwecke müssen wir nur einen kleinen Schritt weiter in die Materie eintauchen und treffen auf die sogenannte Tastatur-Warteschlange, die Speicherzellen 631-640. Darin stehen bis zu zehn Tastendrücke, die bereits getätigt und in der dynamischen Tastaturabfrage identifizert, aber bis dahin noch nicht von GET oder INPUT angefordert worden sind. Die Speicherzellen enthalten jeweils die ASCII-Codes, wobei 631 die nächste, 632 die zweitnächste auszuführende

Taste usw. angibt. Wird ein Zeichen aus dem Tastaturpuffer an GET/INPUT weitergeleitet (»abgefertigt«), so rücken alle folgenden Zeichen wie Menschen in einer Warteschlange auf: 631 verschwindet, der Inhalt von 632 wird zu 631 »befördert«, 633 nach 632 usw. Dadurch entsteht auch regelmäßig Platz für neue Eingabezeichen.
Die Anzahl der Tastendrücke, die sich in der Warteschlange aufhalten, gibt die Adresse 198 an (bewegt sich zwischen 0 und 10). Deshalb löscht

```
POKE 198,0
```

den Tastaturpuffer; in eigenen Programmen ist dies sehr praktisch, wenn man vor einem wichtigen INPUT/GET (Sicherheitsabfrage o.ä.) sicherstellen möchte, daß nur diejenigen Tastendrücke berücksichtigt werden, die nach Erscheinen der INPUT/GET-Aufforderung erfolgten. Analog wartet

```
POKE 198,0:WAIT 198,1
```

solange, bis der Tastaturpuffer wieder aufgefüllt wird, also auf einen beliebigen Tastendruck. Der Nachteil: Nach WAIT ist der Tastaturpuffer-Inhalt nach wie vor erhalten und könnte sich beim nächsten GET/INPUT unnötig melden. Beispiel:

```
100 PRINT "DRUECKEN SIE EINE TASTE:"
110 POKE 198,0:WAIT 198,1
120 GET A$:PRINT A$
```

In Zeile 120 wird also nochmals die Taste berücksichtigt, auf welche Zeile 110 gewartet hat. Somit hebt sich

```
100 POKE 198,0:WAIT 198,1:GET A$
```

von GET A$ dadurch ab, daß der GET-Befehl nie einen Leerstring, sondern immer einen gültigen Tastendruck zurückgibt. Beim C 16 und C 128 heißt dies übrigens GETKEY A$ und funktioniert nach dem gleichen Prinzip wie obige Befehlsfolge.

1.4.3 Anwendungen des Tastaturpuffers

Hört sich an wie ein Widerspruch: Programmierter Direktmodus. Damit ist gemeint, daß sich der Computer innerhalb eines Programms (also aus dem Programm-Modus heraus) kurzfristig in den Eingabezustand (Direktmodus) versetzt, dort Eingaben tätigt, die vorprogrammiert wurden, und nach deren Ausführung ins Programm zurückspringt. Dies ist aufgrund der »dynamischen Tastaturabfrage« durchaus möglich, indem der Tastaturpuffer mit den Codes von Tasten belegt wird, die wir als eingegebene Zeichen vortäuschen wollen. Als erstes wollen wir die Eingabe des Buchstabens A, der den ASCII-Code 65 hat, vortäuschen:

```
POKE 631,ASC("A"):POKE 198,1
oder: POKE 631,65:POKE 198,1
```

Auf diese Befehle hin, die an die vorderste Front des Tastaturpuffers den Buchstaben A schreiben und gleichzeitig die Anzahl der Zeichen im Tastaturpuffer auf 1 setzen (weil ja genau ein Zeichen gesetzt wurde), erscheint

```
READY.
A
```

Beachten Sie den Unterschied zu folgender Eingabefolge, die nichts mit programmiertem Direktmodus zu tun hat:

```
PRINT "A"

A
READY.
```

Zurück zur Simulation von Tastendrücken. Täuschen wir noch schnell zwei Tastendrücke nach gleichem Schema vor:

```
POKE 631,ASC("A"):POKE 632,ASC("B"):POKE 198,2

READY.
AB
```

Auf gleiche Weise versetzen wir das Betriebssystem in den furchtbaren Irrtum, eine Steuertaste sei gedrückt worden. Als Beispiel wollen wir

`<SHIFT CLR/HOME>A<RETURN>` simulieren:

```
POKE 631,147:POKE 632,ASC("A"):POKE 633,13:POKE 198,3
```

Das ergibt unter anderem einen SYNTAX ERROR, weil A zwar über ein simuliertes <RETURN> bestätigt wurde, aber noch lange kein korrekter Basic-Befehl ist. Doch auch die Ausführung von Befehlen ist nicht länger ein Wunschtraum, dazu werden lediglich mehrere Zeichencodes in den Tastaturpuffer geschrieben.

1.4.4 Simulierte Kommandos

Nach dieser vielen Theorie folgt wieder ein Beispiel von der Masterdiskette:

```
LOAD "PROGR.DIREKTMOD.",8
```

lädt folgendes Demonstrationsprogramm (siehe Seite 44):

Listing 1.3: Erstes Programmbeispiel zum programmierten Direktmodus

```
100 REM *******************************
110 REM *                             *
120 REM * DEMONSTRATIONSPROGRAMM FUER *
130 REM *                             *
140 REM * DIE TASTATURPUFFER-NUTZUNG  *
150 REM *                             *
160 REM * ZUM SIMULIERTEN DIREKTMODUS *
170 REM *                             *
180 REM *******************************
190 :
200 PRINT "(CLR)IM MOMENT BEFINDET SICH DER C64 IM"
210 PRINT:PRINT"PROGRAMM-MODUS.                    (RVS)RETURN(OFF)"
220 GET A$:IF A$<>CHR$(13) THEN 220
230 TX$="GOTO 1000"+CHR$(13)
240 FOR X=1 TO LEN(TX$)
250 :POKE 630+X,ASC(MID$(TX$,X))
260 NEXT
270 POKE 198,LEN(TX$)
280 :
290 END:REM ** VORLAEUFIGES PROGRAMMENDE
300 :
310 :
1000 PRINT "(DOWN)(DOWN)(RVS)DIES IST DIE  FORTSETZUNG AB ZEILE 1000."
1010 PRINT"DER BEFEHL  >>GOTO 1000<<  WURDE  ZUERST"
1020 PRINT"IN DEN TASTATURPUFFER  GESCHRIEBEN.  BEI"
1030 PRINT"VERLASSEN DES PROGRAMMS  KAM ER ZUR AUS-"
1040 PRINT"FUEHRUNG. FOLGENDE ZEILE LOESTE DAS VOR-"
1050 PRINT"LAEUFIGE PROGRAMMENDE AUS:"
1060 LIST 290
```

Der Bildschirm gibt schon hinreichende Erklärungen, was eigentlich intern abläuft:
Die Zeilen 200–210 geben die reverse Kopfzeile aus. Zeile 220 wartet darauf, daß Sie <RETURN> drücken, woraufhin 230–270 den Befehl »GOTO 1000« sowie das ausführende <RETURN> in den Tastaturpuffer schreiben. Dort bliebe er völlig unberücksichtigt, würde nicht Zeile 290 das Programm zwischenzeitlich beenden, woraufhin im Direktmodus der Befehl zur Ausführung gelangt – wie wenn er von Hand eingegeben worden wäre. Ab Zeile 1000 erfolgt eine weitere, etwas umfangreichere Textausgabe. Der LIST-Befehl in Zeile 1060 beendet das Programm definitiv.
Eine detaillierte Analyse der Zeilen 230–270 bietet sich an, weil diese Zeilen durchaus ausbaufähig sind:
230: TX$, die Stringvariable des auszuführenden Befehls, wird belegt.
240–260: Die Schleife schreibt den Inhalt von TX$ vollständig in den Tastaturpuffer. Dies geschieht zeichenweise, wobei die Zählvariable X die Nummer des zu schreibenden Bytes ist.
270: Die Länge des Strings TX$ wird als Anzahl der Tasten im Tastaturpuffer festgelegt.
Der große Vorteil des programmierten Direktmodus kommt im recht einfachen Demoprogramm nicht zum Tragen: Der String TX$ kann gewissermaßen in letzter Sekunde modifiziert werden. TX$= "GOTO" +STR$(X) würde also eine GOTO-X-Simulation erlauben. Bei unserer bisherigen Methode müssen wir jedoch einen großen Nachteil bedenken: Mehr als 10 Tastendrücke pro simulierter Eingabe sind nicht zu realisieren, weil damit das Fassungsvermögen des Tastaturpuffers erschöpft ist. Aber Sie wissen ja: Wo ein Wille ist, ist auch ein Weg.

1.4.5 Kommando-Simulation via Bildschirm

Erinnern wir uns an einen großen Vorteil des Full-Screen-Editors: Jeder Befehl, der noch in einer logischen Bildschirmzeile vorhanden ist, kann jederzeit wieder durch <RETURN> in der betreffenden Zeile ausgeführt werden. Dies gilt natürlich auch für Programmzeilen-Eingaben, die dadurch wiederholbar bleiben. Genau dieser Fähigkeit bedienen wir uns nun, um den Tastaturpuffer zu entlasten und dadurch mehr Raum für eigene Anwendungen zu schaffen. Das Programm »PROGR.DIREKTM. 2« demonstriert dies.

Listing 1.4:
Programmbeispiel zum programmierten Direktmodus unter Nutzung des Bildschirms

```
100 REM ********************************
110 REM *                              *
120 REM * VERAENDERTES DEMO-PRG. FUER *
130 REM *                              *
140 REM * DIE TASTATURPUFFER-NUTZUNG   *
150 REM *                              *
160 REM * ZUM SIMULIERTEN DIREKTMODUS *
170 REM *                              *
180 REM ********************************
190 :
200 PRINT "(CLR)IM MOMENT BEFINDET SICH DER C64 IM"
210 PRINT:PRINT"PROGRAMM-MODUS.                    (RVS)RETURN(OFF)"
220 GET A$:IF A$<>CHR$(13) THEN 220
230 TX$="GOTO 1000"
240 PRINT "(CLR)"TX$
250 POKE 631,19:REM *** HOME
260 POKE 632,13:REM *** RETURN
270 POKE 198,2:REM *** 2 TASTENDRUECKE
280 :
290 END:REM ** VORLAEUFIGES PROGRAMMENDE
300 :
310 :
1000 PRINT "(DOWN)(DOWN)(RVS)DIES IST DIE  FORTSETZUNG AB ZEILE 1000."
1010 PRINT"DER BEFEHL >>GOTO 1000<<  WURDE NICHT IN"
1020 PRINT"DEN TASTATURPUFFER  GESCHRIEBEN, SONDERN"
1030 PRINT"IN DIE OBERSTE BILDSCHIRMZEILE. BEI VER-"
1040 PRINT"LASSEN DES PROGRAMMS  WURDE  DIE TASTEN-"
1050 PRINT"FOLGE (RVS)[HOME] [RETURN](OFF)  SIMULIERT, WELCHE"
1060 PRINT"DIE BEFEHLSAUSFUEHRUNG VERANLASSTE."
```

Dabei ändern sich hinsichtlich der Programmtechnik nur die Zeilen 200–270. Diesmal wird der auszuführende Befehl nicht mehr in den Tastaturpuffer geschrieben, sondern auf den Bildschirm gebracht, und zwar in die oberste Zeile. Darauf folgt die Simulation von <HOME> <RETURN>, also nunmehr zwei Tastendrücken. Diese führen den Cursor in die Zeile, in welcher bereits »GOTO 1000« steht, und bestätigen den Befehl.
Die Vorzüge dieser neuen Technik beweist das Beispielprogramm »MEHRERE BEFEHLE«, nach dessen Start Sie auch eine beliebige Taste drücken müssen (ein auszuführender Befehl lautet WAIT 198,1).

Listing 1.5: Simulation mehrerer Befehle im programmierten Direktmodus

```
100 REM ********************************
110 REM *                              *
120 REM *   AUSFUEHRUNG MEHRERER AM    *
130 REM *                              *
140 REM *    BILDSCHIRM  STEHENDER     *
150 REM *                              *
160 REM *   BEFEHLE  IM DIREKTMODUS    *
170 REM *                              *
180 REM ********************************
190 :
200 PRINT "(CLR)IM MOMENT BEFINDET SICH DER C64 IM"
210 PRINT:PRINT"PROGRAMM-MODUS.                    (RVS)RETURN(OFF)"
220 GET A$:IF A$<>CHR$(13) THEN 220
230 PRINT"(CLR)PRINT"CHR$(34)"(RVS)BEFEHL #1"CHR$(34)"
240 PRINT "(DOWN)(DOWN)(DOWN)PRINT "CHR$(34)"(RVS)BEFEHL #2"CHR$(34)
250 PRINT"(DOWN)(DOWN)(DOWN)POKE198,0:WAIT 198,1:POKE 198,0"
260 POKE 631,19:REM *** HOME
270 POKE 632,13:POKE 633,13:POKE 634,13:POKE 198,4:REM *** 3MAL RETURN
280 :
290 SYS 42112:REM *** WARMSTART (UMGEHT AUSGABE VON >>READY.<<)
```

Bemerkenswert ist dabei Zeile 290: SYS 42112 löst einen Warmstart, ähnlich END, aus; das in diesem Zusammenhang lästige »READY.« bleibt jedoch aus.
Die Zeilen 230–250 schreiben drei Befehle auf den Bildschirm, die Zeilen 260–290 bewirken indirekt die Ausführung. Durch folgende Änderung können Sie die Wirkung der Zeilen 230–250 besser erfassen:

```
255 GOTO 255:REM ENDLOSSCHLEIFE
```

Die einzige Schwierigkeit beim Erstellen der PRINT-Zeilen besteht darin, daß man im voraus wissen muß, wo sich der Cursor nach simulierter Ausführung eines Befehls befindet. Im Zweifelsfall hilft dabei nur eines: Ausprobieren, bis sich die Anzahl der CURSOR DOWNs richtig einpendelt.

1.4.6 Selbstmodifikation von Basic-Programmen

Eine persönliche Vorbemerkung: Wenn ich in diesem Abschnitt von der »Selbstmodifikation« spreche, so handelt es sich dabei um eines meiner Lieblingsthemen. Entsprechende Techniken führe ich in Kapitel 10 auch in Maschinensprache aus.
Zurück zum Begriff. Unter »Modifikation« versteht man eine »Veränderung« eines bestehenden Programms, zum Beispiel die Einfügung der Endlosschleife (Zeile 255) im vorausgegangenen Unterkapitel 1.4.5. Auch Spiele-Pokes sind Programm-Modifikationen. Im folgenden werde ich das Wort »Modifikation« anstelle von »Manipulation«, welches einen eher negativen Beigeschmack hat, gebrauchen.
Ist nun ein Programm in der Lage, »sich selbst zu modifizieren«, so bedeutet dies, daß das Programm während seines Ablaufes an sich selbst programmierte Veränderungen vornimmt. Diese bestehen im Entfernen, Anpassen oder Einfügen von Befehlen und Befehlszeilen. Dadurch

erschließt man sich zahlreiche Möglichkeiten, schwierige Problemstellungen bestmöglich zu lösen. Ein Beispiel: Wenn ein Programm eine beliebige Funktion mit DEF FN so definieren möchte, daß sie vom Anwender eingegeben werden kann, ist dies im normalen Basic nicht zu realisieren. Doch unter Zuhilfenahme der Selbstmodifikation meistert das Beispielprogramm »ZEILENGENERATOR« auch dieses Problem; laden und starten Sie es bitte. Sie sollen dann eine Funktion von X, zum Beispiel X*5, eingeben, sowie einen Testwert, für den dann die Berechnung des Funktionswertes erfolgt. Beispiel:

```
Y(X) = X*50
X    = 3
Y=F(X)=X*50=150
```

Zulässig ist dabei die gesamte Basic-Syntax für DEF FN, also ein riesiges Spektrum an Funktionen und Operatoren. Listen Sie jetzt die Zeilen ab 1000; dort stehen dann hinter der Funktionsdefinition ganz normale Anweisungen (INPUT und PRINT). Der entscheidende Trick besteht jedoch in der Modifikation der Zeilen 1010 und 1020:

Listing 1.6: Selbstmodifikation in Basic

```
100 REM ******************************
110 REM *                            *
120 REM *   SELBSTMODIFIKATION VON   *
130 REM *                            *
140 REM *     BASIC-PROGRAMMEN       *
150 REM *                            *
160 REM *   DURCH SIMULATION EINER   *
170 REM *                            *
180 REM *   DIREKTEN ZEILENEINGABE   *
190 REM *                            *
200 REM ******************************
210 :
220 :
230 Z1=1010
300 PRINT "(CLR)(DOWN)(DOWN)WELCHE FUNKTION ?"
310 INPUT "Y=F(X)";F$
320 PRINT "(CLR)(DOWN)"
330 PRINT Z1;"F$=";CHR$(34);F$;CHR$(34)
340 PRINT Z1+10;"DEF FN F(X)=";F$
350 PRINT "RUN";Z1-10;CHR$(19);
360 POKE 631,13:POKE 632,13:POKE 633,13:POKE 198,3:END
1000 REM *** FORTSETZUNG ***
1010 REM // DIESE ZEILE WIRD MODIFIZIERT
1020 REM // DIESE ZEILE AUCH !
1030 PRINT "(DOWN)(DOWN)FUNKTION: Y(X)="F$
1040 INPUT "X-WERT";X
1050 PRINT "Y=F(X)=";F$;"=(RVS)";FN F(X)
```

Sowohl der eingegebene Funktionsstring F$ als auch die Funktionsdefinition selbst landen in diesen beiden Zeilen. Da der Basic-Interpreter dieses Vorgehen akzeptiert, gelingt es uns somit, eine Anweisung wie INPUT FN F(X) zu simulieren. Auf diese Weise gehen übrigens alle Funktionskurven-Plotter-Programme, die in Basic geschrieben sind, vor.

Zurück zu unserem Beispielprogramm. Als Listing 1.6 sehen Sie dessen »Urzustand«; so wird das Programm von Diskette geladen, so sieht es vor (!) dem ersten Start aus. Beachten Sie die Zeilen 1010/1020, die momentan noch REM-Anweisungen beinhalten. Starten Sie nun das Programm und geben Sie als Funktion »5-X+3*SIN(X)«, als X-Wert einen beliebigen Wert ein. Listen Sie anschließend die Zeilen 1010/1020, so erhalten Sie – man lese und staune – folgenden Inhalt:

```
1010 F$="5-X+3*SIN(X)"
1020 DEF FN F(X)=5-X+3*SIN(X)
```

Die Stringdefinition in Zeile 1010 geschieht nur, um die Variable F$ für 1030 und 1050 zu retten; bei der simulierten Zeileneingabe gehen nämlich alle Variablen verloren, so daß man die wichtigen Werte wiederum über den Bildschirmspeicher aufheben und neu definieren muß.
Nun zum Ablauf des Beispielprogramms. Die Konstante Z1 erhält in Zeile 230 die Nummer der ersten Modifikationszeile (1010). In 300/310 holt sich das Programm den Definitionsstring für die Funktion nach F$. Nun werden auf den Bildschirm die beiden Zeileneingaben in 330/340 sowie der RUN-Befehl in 350 geschrieben; dies am Bildschirm zu verfolgen, dürfte kein Problem sein. Die POKE-Anweisungen aus Zeile 360 lösen ein dreifaches <RETURN> (zwei Zeileneingaben, ein RUN-Befehl) aus und beenden das Programm; der Cursor befindet sich zu diesem Zeitpunkt bereits in der obersten Bildschirmzeile, weil in Zeile 350 die HOME-Positionierung über CHR$(19) erfolgte. Der END-Befehl führt nun temporär in den Eingabemodus, wo die Zeileneingaben und der RUN-Befehl zur Ausführung kommen. Die Fortsetzung ab Zeile 1000 beginnt dann mit den beiden modifizierten Zeilen, wodurch F$ wieder seinen alten Wert erhält und die Funktion FN F(X) definiert wird. Die restlichen drei Zeilen geben noch einmal die Funktionsdefinition aus F$ aus (1030), lesen einen X-Wert zur Berechnung ein (1040) und drucken Funktionsdefinition sowie Funktionswert (1050).

Ausblicke

Nach dem beschriebenen Verfahren lassen sich geniale Lösungen für die unterschiedlichsten Programmierzwecke entwickeln; es kommt jedoch jeweils auf die entsprechende Situation an, inwiefern die Selbstmodifikation zum gewünschten Ergebnis führt. Es sei jedoch auch erwähnt, daß sogar kleinere Programmgeneratoren, also Programme, die ihrerseits wieder Programme erzeugen, nach der beschriebenen Methode realisierbar sind. Der eigentliche Generator steht dazu zweckmäßigerweise in höheren Programmzeilen, während der generierte Code nach und nach in den Anfangszeilen entsteht. Der eigentliche Generator löscht sich schließlich stückweise, indem er jeweils die Nummern der zu entfernenden Zeilen auf den Bildschirm ausgibt und durch simulierte RETURNs schließlich löschen läßt; nach jedem Satz von maximal 9 Lösch-Zeilennummern folgt die Neubelegung der wichtigsten Variablen sowie der Rücksprung in die Lösch-Schleife, welche dann ihre Löschtätigkeit fortsetzt.
Eine weitere Idee wäre ein Sprite-Editor, der nicht nur die DATA-Werte eines Sprites berechnet, sondern diese auch gleich in einem gewünschten Zeilenbereich unterbringt; in Kapitel 2 erhalten Sie das Grundgerüst für einen Sprite-Editor, der sich dahingehend weiterentwickeln läßt.

Eine andere Anwendung bestünde in einem Maskengenerator (Programm, das Bildschirminhalte in fertige PRINT-Zeilen umwandelt); einen solchen erhalten Sie in Kapitel 9, allerdings als Maschinenprogramm.

1.4.7 Diverse Tricks zum Tastaturpuffer

Eigenes INPUT-Zeichen

Das Fragezeichen, welches bei jeder INPUT-Aufforderung erscheint, ist in vielen Fällen sehr störend. Über einen kleinen Trick läßt sich jedoch ein eigenes INPUT-Zeichen definieren, indem die folgenden Tastendrücke vorgetäuscht werden:

```
<DEL> <DEL> <neues Zeichen> <Leerzeichen>
```

Diese Beispielzeile gibt ein Größer-Symbol anstelle des Fragezeichens aus:

```
10 POKE 631,20:POKE 632,20:POKE 633,ASC(">"):POKE 634,32:POKE 198,4:
INPUT A$
```

AUTO-Befehl in Basic

Mit diesem Mini-Programm lassen sich Zeilennummern automatisch vorgeben. Gestartet wird das Programm mit

```
A=Anfangszeilennummer:GOTO 1
```

Hier das Programm, welches sich nicht auf der Masterdiskette befindet, weil es bei jeder Verwendung neu eingetippt werden muß:

```
1 PRINT"<SHIFT CLR/HOME>"A;:POKE 19,1:INPUT A$:POKE 19,0:PRINT
2 PRINT"A="A+10":GOTO1":POKE 198,3:POKE 631,19:POKE 632,13:POKE 633,13
:END
```

Nun wird der Bildschirm gelöscht und die in A definierte Zeilennummer ausgegeben. Mit POKE 19,1 wird die Tastatur zum aktuellen Eingabegerät. Dadurch wird bei der nachfolgenden INPUT-Anweisung das Fragezeichen unterdrückt. Nach der Eingabe der Zeile erscheint darunter:

```
A=(nächste Zeilennummer):GOTO 1
```

In den Tastaturpuffer wird <HOME> und <RETURN> geschrieben, so daß die Zeile nach dem END-Befehl automatisch ins Programm integriert wird. Die Schrittweite ist normalerweise 10 und steht am Anfang von Zeile 2; sie läßt sich verändern.
Das Programm kann nur mit <RUN/STOP RESTORE> verlassen werden.

Raffinierter ON ERROR GOTO

Wer dringend einen »ON ERROR GOTO«-Befehl benötigt und keine entsprechende Erweiterung hat, kann das mit ein paar POKE-Anweisungen simulieren. »ON ERROR GOTO« heißt, daß bei Auftreten irgendeines Programmfehlers wie SYNTAX ERROR nicht das Programmende, sondern eine eigens vom Programm bereitgestellte Fehlerbehandlung aktiv wird. Es werden ganz einfach die CHR$-Codes des Wortes GOTO in den Tastaturpuffer ab 631 gePOKEt, anschließend die Werte der Zeilennummer und dann noch CHR$(13) für <RETURN>. Zuletzt kommt in Speicherzelle 198 noch die Anzahl der in den Tastaturpuffer geschriebenen Werte. Dies funktioniert natürlich nicht bei Tastaturabfragen während des Programms, es sei denn, man initialisiert den Tastaturpuffer danach jedesmal erneut, z.B. in einer Unterroutine.

Abbildung 1.1: Das Listing zu PRINT-LIST

```
10 REM ********************************         <217>
20 REM * REM-ZEILEN NICHT ABTIPPEN !! *         <218>
30 REM ********************************         <237>
40 :                                            <098>
210 REM ********************************        <119>
240 REM *        PETER       ZUSER      *       <236>
250 REM *        BAHNHOFSTR.  346       *       <151>
260 REM *        A-8950  STAINACH       *       <119>
270 REM *        ================       *       <194>
290 REM *        TEL.: 03682/2648       *       <089>
300 REM ********************************        <210>
310 :                                           <113>
63974 PRINT CHR$(147)CHR$(17)                   <172>
63975 PRINT TAB(11)"PROGRAMM PRINTLIST"         <025>
63976 PRINT TAB(11)"=================="         <054>
63977 PRINT                                     <123>

63979 PRINT TAB(12)"ZUSER PETER 1984"           <223>
63980 FOR K=1 TO 4:PRINT:NEXT                   <251>
63981 INPUT"UEBERSCHRIFT: ";UE$                 <104>
63982 AN=PEEK(43)+256*PEEK(44)                  <016>
63983 ZN=-1:LZ=63974:SZ=0                       <134>
63984 OPEN 1,4:PRINT#1                          <028>
63985 PRINT#1,UE$                               <157>
63986 SZ=SZ+1:EZ=ZN+1                           <224>
63987 FOR I=1 TO 60:ZN=PEEK(AN+2)+256*PEEK
      (AN+3)                                    <039>
63988 IF ZN=LZ THEN ZN=ZN-1:ZZ=I:I=61           <130>
63989 AN=PEEK(AN)+256*PEEK(AN+1)                <065>
63990 NEXT                                      <113>
63991 PRINT#1,SPC(70)"SEITE:";SZ                <251>
63992 POKE 198,10                               <183>
63993 FOR J=631 TO 640:READ A:POKE J,A:NEX
      T:RESTORE                                 <083>
63994 PRINT CHR$(147)"CMD1:LIST"EZ"-"ZN:EN
      D                                         <113>
63995 FOR K=1 TO 4:PRINT#1:NEXT                 <058>
63996 ON I-60 GOTO 63985                        <120>
63997 FOR L=1 TO 61-ZZ:PRINT#1:NEXT             <207>
63998 CLOSE 1:POKE 198,0:END                    <081>
63999 DATA 19,13,71,207,54,51,57,57,53,13       <073>
```

Dieses Verfahren kann man natürlich auch zu anderen Zwecken benutzen, z.B. für ein automatisches LIST bei einem Fehler oder Programmende. Dies entspricht dem in Listing 1.3 gezeigten Trick; ändern Sie den dortigen END-Befehl (Zeile 290) in eine Anweisung, die einen Fehler auslöst (z.B. »A=B/0« oder »;« o.ä.).

Eigene LIST-Gestaltung

Der LIST-Befehl ist in der Grundform relativ unflexibel. So druckt er beispielsweise bei Endlospapier über die Perforation, und die Seitennumerierung fehlt ihm ebenfalls. In der Ausgabe 4/85 des 64'er-Magazins wurde dazu ein Programm namens »PRINT-LIST« auf Seite 79 vorgestellt, welches diese Fähigkeiten hat. Abbildung 1.1 ist das Listing zu dieser Routine, welches lediglich als Anregung dient und nicht auf der Masterdiskette steht.

Tastatur außer Gefecht setzen

Die Tastatur wird durch folgenden Befehl abgeschaltet:

```
POKE 649,0
```

Damit wird der sogenannte XMAX-Speicher, welcher die maximale Kapazität des Tastaturpuffers angibt, auf 0 gesetzt. Infolgedessen bricht die gesamte Tastaturbehandlung zusammen, weil augenscheinlich kein freier Speicherplatz zum Vermerken von Tastendrücken vorhanden ist. Den Ausgangszustand reaktiviert

```
POKE 649,10
```

1.5 Die Funktionstasten

Ein Problem für sich stellt die Behandlung der Funktionstasten dar, welche zwar in der Werbung als großer Vorzug gepriesen, normalerweise jedoch kaum genutzt werden. Dennoch gibt es zahlreiche Programme (auch auf der Masterdiskette dieses Buches), die die Funktionstasten mit immer wiederkehrenden Zeichenfolgen belegen. Dadurch verringert sich die Tipparbeit bei Anwendung und Programmierung.
Dieser Abschnitt stellt kurz die Behandlung der Funktionstasten im Normalzustand sowie deren Belegung mit eigenen Texten vor.

a. Abfrage der Funktionstasten

Die Funktionstasten werden wie Steuertasten behandelt, d.h., sie haben im Normal Mode keine Funktion, ergeben aber im Insert Mode und Quote Mode reverse Symbole bzw. Großbuchstaben im Kleinschriftmodus.

Folgende Steuercodes sind den Funktionstasten zugewiesen:

F1 CHR$(133)
F2 CHR$(137)
F3 CHR$(134)
F4 CHR$(138)
F5 CHR$(135)
F6 CHR$(139)
F7 CHR$(136)
F8 CHR$(140)

Etwas unangenehm ist diese Verteilung der Codes, weil die Nummer der Funktionstaste nicht einfach zu einem Ausgangswert (wie 133) addiert werden kann; die ungeshifteten Funktionstasten, also diejenigen mit den ungeraden Nummern, belegen die Codes 133–136, die geshifteten (geradzahlige Nummern) die Werte 137–140.

Bemerkenswert ist allgemein die Tatsache, daß die Steuercodes der Funktionstasten keinerlei Wirkung haben. Deshalb fällt deren Auslösung im Normal Mode auch höchstens durch ein kurzfristiges Stocken des Cursorblinkens auf, nicht jedoch als Steuerfunktion. Die Funktionstasten-Codes dienen nämlich ausschließlich der Abfrage dieser Tasten, zum Beispiel in Menüs. Folgendes Beispielprogramm (nicht auf der Masterdiskette enthalten) wartet auf das Auslösen einer Funktionstaste und nennt daraufhin deren Bezeichnung; es ist verhältnismäßig umständlich geschrieben, um möglichst viele Beispielabfragen zu enthalten:

```
100 poke 198,0:wait 198,1:get a$
110 IF A$=CHR$(133) THEN PRINT "F1"
120 IF A$=CHR$(137) THEN PRINT "F2"
130 IF A$=CHR$(134) THEN PRINT "F3"
140 IF A$=CHR$(138) THEN PRINT "F4"
150 IF A$=CHR$(135) THEN PRINT "F5"
160 IF A$=CHR$(139) THEN PRINT "F6"
170 IF A$=CHR$(136) THEN PRINT "F7"
180 IF A$=CHR$(140) THEN PRINT "F8"
190 END
```

Die Eingabe von CHR$(x) können Sie durch den Quote Mode bequem umgehen. Beispiel:

```
110 if a$="<f1>"then print "f1"
```

Der Vorteil liegt dabei vor allem darin, daß Sie nicht umständlicherweise den Tastencode aus einer ASCII-Tabelle herausklauben müssen.

b. Belegung der Funktionstasten

Die Frage, wie man die Funktionstasten mit eigenen Zeichenketten belegt, wurde schon tausendfach von anderen Autoren beantwortet. Deshalb wollte ich nicht »irgendein x-beliebiges« Programm zu diesem Thema veröffentlichen, sondern habe mir das bislang beste seines Genres herausgesucht. Es wird von der Masterdiskette durch

```
load "key-32",8
```

geladen und mit Run gestartet. Den Eingabetrick mit <Shift Run/Stop> möchte ich nur am Rand erwähnen (siehe 1.1).

Wie der Name schon sagt: Mit dem Programm »KEY-32« können Sie insgesamt 32 Texte mit je 31 Zeichen über die Funktionstasten abrufen. Nun werden sich manche sicher fragen, wie man denn mit vier Tasten (<F1>, <F3>, <F5> und <F7>) 32 Funktionen abrufen soll. Diese enorme Vielzahl wird durch Kombination der vier Funktionstasten des C64 mit den Tasten <SHIFT>, <C=> und <CTRL> erreicht. Am Anfang wird das Basic-Programm »KEY-32« geladen, welches das eigentliche Maschinenprogramm »M-KEY-32« automatisch nachlädt. Nun erscheint die Eingabemaske mit dem komfortablen Eingabefeld. Die Eingabefolge wird auf dem Bildschirm erklärt. Hier jedoch zusätzlich einige Erläuterungen:

Zuerst wird eine Zahl von 1 bis 4 eingegeben:

1	für	<F1>
2	für	<F3>
3	für	<F5>
4	für	<F7>

Die Tasten <F2>, <F4>, <F6> und <F7> bleiben zunächst unberücksichtigt, da diese durch Kombination der genannten vier mit <SHIFT> entstehen. Danach wird deshalb eine Zahl von 0 bis 7 für die Sondertasten angegeben:

0	für	<keine Sondertaste>
1	für	<SHIFT>
2	für	<C=>
3	für	<C=> + <SHIFT>
4	für	<CTRL>
5	für	<CTRL> + <SHIFT>
6	für	<CTRL> + <C=>
7	für	<CTRL> + <C=> + <SHIFT>

Will man zum Beispiel eine Belegung der Tastenkombination <F3> + <C=> + <SHIFT> erreichen, so gibt man die Ziffern 2 (für F3> und 3 (für <C=> + <SHIFT>) ein. Dieses Feld mit den beiden Zahlen wir revers dargestellt. Eine eventuelle Fehleingabe wird vom sehr bedienungssicheren Programm verhindert.

Nun gibt es zwei Möglichkeiten. Entweder gibt man direkt nach den zwei Zahlen einen Klammeraffen ein, oder man tippt für diese Tastenkombination den Begleittext mit anschließendem Klammeraffen ein. Ersteres listet die Tastenbelegung; man kann also sehen, ob und womit die Taste belegt ist. Die zweite Möglichkeit speichert den eingegebenen Text mit der definierten Tastenkombination.

Beispiel: Belegung der F1-Taste ohne Sondertaste.
Belegen: 10load"$",8<@>
Listen: 10<@>

Mit der Cursortaste bewegt man sich im Eingabefeld. Der <←> löscht die angezeigte Eingabe, nicht aber die schon gespeicherten Tastenbelegungen und setzt den Cursor an den Text-

beginn. Will man die gesamte Tastenbelegung betrachten, drückt man <£>. Dies zeigt aus Platzgründen nur die ersten 16 Tasten. Nach Auslösen von <2> kann man das zweite Blatt mit den Tasten 17–32 betrachten. Das Tippen der Taste <H> führt uns wieder ins Hauptmenü, die Eingabemaske.
Will man die fertige Tastenbelegung für späteren Gebrauch auf Diskette speichern, so verwendet man <SHIFT> + <£>. Dadurch wird das eigentliche Maschinenprogramm »M-KEY-32« erzeugt. Der Druck auf die Tasten <CTRL> + <←> ruft dieses Programm auf. Die neue Bildschirmanzeige und der Piepston bei jedem Tastendruck lassen die Funktionsbereitschaft erkennen.

Hinweise zum Maschinenprogramm

Das Maschinenprogramm »M-KEY-32« wird mit LOAD"M-KEY-32",8,1 geladen und nach NEW mit SYS 52000 gestartet. Es belegt den Speicher von $cb20 bis $cfff; im Bereich $c000 bis $cfff sind die Inhalte der einzelnen Tasten abgelegt.

Programmhinweise zum Basic-Teil

Die Programmfunktionen werden durch die REM-Anweisungen im Listing ausführlich beschrieben. Die Variablen tragen folgende Bedeutung:

A$,B$	Ein- und Ausgabevariablen
SP	Cursorspalte
CR	gibt die zur Eingabe anstehende Bildschirmposition an
FR	Funktionstastenrubrik (-spalte)
SR	Sondertastenrubrik (-spalte)
I	Schleifenzähler
T1,T2	die ersten zwei Werte im Eingabefeld (F+S)
KF	Klammeraffenabfrage im dritten Eingabefeld
FZ,SZ	Funktionstastenanzahl (-wert), Sondertasten (-zahl)
PB	Basisadresse des Tastentextes für PEEK/POKE
PE	Endadresse des Tastentextes für PEEK/POKE
ZE	Zeichen in dieser Adresse
FL	Flag für Textende
SA	Startadresse (-position) des Ausgabetextes
LO	Schleife für 31 einzelne Zeichen des Ausgabetextes
HI	Schleife für 16 verschiedene Tasten
PS	POKE-Stelle (-position)
PW	PEEK/POKE-Wert

Listing 1.7: Das Basic-Steuerprogramm zu KEY-32

```
10 REM      --- K E Y   3 2 ---
20 REM COPYRIGHT: SIEGBERT WERNER
30 REM BEETHOVENSTR. 59 SIEGEN 31
40 POKE53280,0:POKE53281,0:POKE646,15
42 A=A+1
44 IFA<2THENLOAD"M-KEY-32",8,1
50 SP=3:REM-------CURSOR IN SPALTE 3
60 GOSUB580:REM BILDSCHIRMAUFBAU
70 GOTO190:REM----CURSOR SETZEN
78 REM------------TASTATURABFRAGE
80 POKE198,0:WAIT198,1
90 GETA$
97 REM------------UNERWUENSCHTE TASTE?
98 REM------------Z.B.:RETURN,HOME,ECT.
99 REM------------DANN NICHT REAGIEREN
100 IFASC(A$)=17ORASC(A$)=145ORASC(A$)=148ORASC(A$)=20ORASC(A$)=147ORASC(A$)=19T
HEN190
105 IFASC(A$)=13THEN190
107 REM------------ABFRAGE DER ERLAUBTEN
108 REM------------TASTEN
109 REM------------CURSOR LINKS-TASTE?
110 IFASC(A$)=157THENSP=SP-2
120 IFSP<2THENSP=2:REM-CURSORFELDANFANG
129 REM------------CTRL MIT ← TASTE?
130 IFASC(A$)=6THENGOTO340  :REM PRG-START!
135 IFA$="£"THEN880
137 IFA$="◤"THEN1810
140 IFA$="←"THENGOTO290
150 PRINTA$
160 IFA$="@"THENGOSUB400
170 IFSP=36THEN190:REM-CURSORFELDENDE
180 SP=SP+1:REM---CURSOR EINS RECHTS
189 REM------------CURSOR SETZEN
190 POKE211,SP
200 POKE214,23
210 SYS58732
218 REM------------SETZEN DES ZEIGERS
219 REM------------AUF DAS EINGABEFELD
220 CR=1024+(PEEK(214)+1)*40+PEEK(211)
230 POKECR,30:POKECR-1,67:POKECR+1,67
238 REM------------INVERTIEREN DER
239 REM------------ZAHLEN FUER F+STASTE
240 FR=PEEK(1947):SR=PEEK(1948)
250 IFFR<=127THENFR=FR+128
260 IFSR<=127THENSR=SR+128
270 POKE1947,FR:POKE1948,SR
279 REM------------NEUE TASTE ABFRAGEN
280 GOTO80
289 REM------------EINGABEFELD LOESCHEN
290 POKECR,67:POKECR-1,67:POKECR+1,67
300 SP=3
310 A$=""
320 FORI=0TO33:POKE1947+I,32:NEXTI
330 GOTO190
339 REM------------MASCHINENPRG.-START!
340 :A$=" "
350 SYS 52000:REM-IRQ AUF NEUE ROUTINE!
359 REM------------BEREITSCHAFTSBILD
360 PRINT"(CLR)"
370 PRINT"              K E Y   3 2          "
380 PRINT"----------------------------------------";
382 POKE211,0:POKE214,23:SYS58732
384 PRINT"----------------------------------------";
```

```
388 POKE211,0:POKE214,2:SYS58732
390 END
398 REM------------FUER F+S NUR ZAHLEN
399 REM------------ERLAUBEN!
400 T1=PEEK(1947)-128:T2=PEEK(1948)-128
402 IFT1<49ORT1>52THEN840
404 IFT2<48ORT2>55THEN840:REM-FEHLER
408 REM------------SPEICHERN? - ZEIGEN?
410 KF=PEEK(1949)
420 IFKF<>0THEN510
429 REM------------KEYBELEGUNG ZEIGEN
430 FZ=PEEK(1947):SZ=PEEK(1948)
440 PB=51968+(FZ-176)*256+(SZ-176)*32
450 FORI=0TO31:PE=PB+I:ZE=PEEK(PE)
460 IFZE=0THEN490:REM-TEXTENDE=@=0
470 POKE1949+I,ZE:REM-ZEICHENAUSGABE
480 NEXTI
490 SP=4:RETURN
500 REM------------KEYBELEG. SPEICHERN
510 FZ=PEEK(1947):SZ=PEEK(1948)
520 PB=51968+(FZ-176)*256+(SZ-176)*32
530 FORI=0TO31:PE=1949+I:ZE=PEEK(PE)
540 POKEPB+I,ZE
550 IFZE=0THEN290
560 NEXTI
570 GOTO290
579 REM------------BILDSCHIRMAUFBAU
580 PRINT"(CLR)(RVS)                                        ";
590 PRINT"(RVS)            K E Y   3 2                ";
600 PRINT"(RVS)                                       (OFF)";
610 PRINT
620 PRINT" ╭────────╮ ╭──────────────────────────╮ ";
630 PRINT" │F-TASTEN│ │S-TASTEN                  │ ";
640 PRINT" │=       │ │=                         │ ";
650 PRINT" │1= F1+F2│ │0= OHNE SONDERTASTE       │ ";
660 PRINT" │2= F3+F4│ │1= SHIFT                  │ ";
670 PRINT" │3= F5+F6│ │2= COMMODORE              │ ";
680 PRINT" │4= F7+F8│ │3= SHIFT+COMMODORE        │ ";
690 PRINT" ╰─┬──────╯ │4= CTRL                   │ ";
700 PRINT"   │        │5= SHIFT+CTRL             │ ";
710 PRINT"   │        │6= COMMODORE+CTRL         │ ";
720 PRINT"   │ ╭──────┤7= SHIFT+COMMODORE+CTRL   │ ";
730 PRINT"  F│        ╰──────────────────────────╯ ";
740 PRINT"   │S       ╭──────────────────────────╮ ";
750 PRINT"   ││ ╭─────┤TEXTEINGABE MIT @ BEENDEN│ ";
760 PRINT"   │││      │NUR @ HINTER F+S= AUSGABE│ ";
770 PRINT"   │││      │CTRL/← =START  £= KEYLIST│ ";
780 PRINT"   │││      │SHIFT/£=SAVEN  ←= LOESCH.│ ";
785 PRINT"   │││      ╰──────────────────────────╯ ";
790 PRINT" ╭─┼┼┼─────────────────────────────────╮"
800 PRINT" -                                     -"
810 PRINT" ╰─────────────────────────────────────╯";
820 RETURN
830 REM---------------EINGABEFEHLER
831 REM---------------KENNTLICH MACHEN!
840 FORI=0TO31:READQ:POKE1949+I,Q:NEXTI
845 POKECR,67:POKECR-1,67:POKECR+1,67
850 RESTORE:SP=3:GOTO190
860 DATA14,21,18,32,6,5,19,20,7,5,12,5,7,20,5,32,26,1,8,12,5,14
870 DATA32,9,14,32,6,43,19,33,32,32
879 REM---------------KEYLISTING BLATT1
880 GOSUB900:SA=52224:GOSUB1600
890 GOTO1360
899 REM---------------KEYLISTING BILD
900 PRINT"(CLR)(RVS)                                        ";
910 PRINT"(RVS)  K E Y   3 2   -BLATT 1-  WERNER-MADE  ";
```

```
920 PRINT"(RVS)                                          (OFF)";
930 PRINT
940 PRINT"     ┌──SHIFT     1 / 2=SEITENWAHL"
950 PRINT"     │┌──COMMODORE H    =RUECKSPRUNG"
960 PRINT"     ││┌─CTRL      CTRL/←=START HAUPTPRG.";
970 PRINT"     │││"
1000 PRINT" F1+○○○=..............................."
1010 PRINT" F1+●○○=..............................."
1020 PRINT" F1+○●○=..............................."
1030 PRINT" F1+●●○=..............................."
1040 PRINT" F1+○○●=..............................."
1050 PRINT" F1+●○●=..............................."
1060 PRINT" F1+○●●=..............................."
1070 PRINT" F1+●●●=..............................."
1080 PRINT" F2+○○○=..............................."
1090 PRINT" F2+●○○=..............................."
1100 PRINT" F2+○●○=..............................."
1110 PRINT" F2+●●○=..............................."
1120 PRINT" F2+○○●=..............................."
1130 PRINT" F2+●○●=..............................."
1140 PRINT" F2+○●●=..............................."
1150 PRINT" F2+●●●=...............................";
1152 RETURN
1350 REM------------WELCHE AUSWAHLTASTE?
1360 POKE198,0:WAIT198,1
1370 GETB$
1380 IFASC(B$)=6THEN340
1390 IFB$="H"THEN50
1400 IFB$="1"THEN880
1405 IFB$="2"THEN1500
1410 GOTO1360
1498 REM------------KEYLISTING BLATT2
1499 REM------------AENDERUNG D. BILDES
1500 POKE1087,178
1502 FORI=0TO7:POKE1346+I*40,53:NEXTI
1504 FORI=0TO7:POKE1666+I*40,55:NEXTI
1507 REM------------LESEADRESSE AENDERN
1508 SA=52736:GOSUB1600
1510 GOTO1360
1599 REM------------AUSGABESCHLEIFE
1600 FL=1:FORHI=0TO15
1610 FORLO=0TO30
1620 PW=PEEK(SA+HI*32+LO)
1630 PS=1352+HI*40+LO
1640 IFPW=0THENFL=0
1650 IFFL=0THENPW=46:REM PUNKTE NACH TEXT
1660 POKEPS,PW
1670 NEXTLO
1680 FL=1
1690 NEXTHI
1700 RETURN
1800 REM            SAVEN DER BELEGUNG
1810 OPEN1,8,1,"@:M-KEY-32"
1820 A$=CHR$(32):B$=CHR$(203)
1830 PRINT#1,A$;B$;
1840 FORI=52000TO53248:A$=CHR$(PEEK(I)):PRINT#1,A$;:NEXTI
1950 CLOSE1
1960 GOTO50:REM------ZURUECK ZUM PRG.
```

2

Mit dem Begriff »Einzeiler« ist der Inhalt dieses Kapitels fast schon zu eng beschrieben; denn neben mehr als 60 Einzeilern, der ausführlichsten Zusammenstellung dieser Art, und einigen Mehrzeilern, erfahren Sie zahlreiche Grundlagen über Funktionsdefinitionen, die Einbindung von Maschinenroutinen in Basic-Zeilen, PEEK/POKE/SYS sowie als krönender Abschluß die mathematische Auswertung logischer Ausdrücke, hinter denen sich besonders interessante und effiziente Programmiertechniken verbergen.

2.1 60 Einzeiler für jeden Zweck

Die Idee des Einzeilers ging von der Redaktion der Zeitschrift 64'er aus, die bereits 1984 einen Wettbewerb zu diesem Thema ausrief. Dabei galt es, innerhalb einer einzigen Programmzeile möglichst viele Effekte zu verpacken. Auf den ersten Blick mag es unmöglich scheinen, mit einer einzigen Zeile anspruchsvolle Problemlösungen zu entwickeln, doch wenn Sie die nun folgenden Einzeiler ansehen, werden Sie aus dem Staunen nicht mehr heraus kommen.

Einzeiler #1: MERGE in Basic

Besonders wenn man sich der vielgepriesenen modularen Programmierung bedient, gerät man oft in die Situation, zwei oder mehr Basic-Programme zu einem einzigen Programm zu verbinden. Hierfür wurden schon zahlreiche Lösungen entwickelt, vor allem in Maschinensprache. Am einfachsten ist jedoch folgender Einzeiler, der in jedes Programm leicht einzufügen ist, an welches ein weiteres Programm(-modul) angehängt werden soll:

Listing 2.1: EZ MERGE

```
10 A=PEEK(45)+256*PEEK(46)-2:POKE44,A/256:POKE43,A-PEEK(44)*256:PRINT"PRG LADEN
& PΓ43,1:PΓ44,8
```

Er berechnet unter Berücksichtigung des Programmende-Zeigers 45/46 nach A die Anfangsadresse, an welche das neu anzuhängende Programm geladen werden muß, damit es genau am Ende des bereits im Speicher befindlichen Programms landet. Diese Adresse wird dann in den

Programmanfangszeiger 43/44 geschrieben (Aufspaltung in Low-/High-Format) und anschließend die Aufforderung ausgegeben, das neu anzuhängende Programm zu laden und anschließend die Befehlsfolge

```
POKE 43,1:POKE 44,8
```

einzugeben (dadurch wird der alte Programmanfang wiederhergestellt). Danach steht das Ergebnis der Verknüpfung im Speicher und kann gespeichert werden.

Einzeiler #2: OLD in Basic

Oftmals wird ein Programm wider Willen oder unabsichtlich durch NEW oder einen Reset gelöscht. Bleibt der C64 jedoch angeschaltet, ist das Programm noch nicht verloren. Folgende Eingabe ist ohne Zeilennummer (!) im Direktmodus zu tätigen und führt zum gewünschten Ergebnis:

Listing 2.2: EZ OLD

```
1 REM *** IM DIREKTMODUS EINZUGEBEN ***
2 :
3 POKE2050,8:SYS42291:POKE46,PEEK(35)-(PEEK(781)>253):POKE45,PEEK(781)+2AND255:C
LR

READY.


1 REM *** IM DIREKTMODUS EINZUGEBEN ***
2

3 POKE2050,8
  SYS42291
  POKE46,PEEK(35)-(PEEK(781)>253)
  POKE45,PEEK(781)+2AND255
  CLR
```

Vorher dürfen jedoch keine Variablen definiert oder Basic-Zeilen eingetippt werden, da sonst das gelöschte, aber noch im Speicher stehende Programm zerstört würde. Das Prinzip des OLD-Einzeilers beruht auf dem Aufruf einer System-Routine, die die Basic-Zeilen neu bindet und dabei als »Abfallprodukt« das Ende des gelöschten Programms herausfindet. Der erste POKE dient nur dazu, der Routine vorzutäuschen, daß sich ein noch ungelöschtes Programm im Speicher befindet (sonst arbeitet sie nicht). Die Endadresse des Basic-Programms wird schließlich um 2 erhöht und in die Adressen 45/46, den Zeiger auf das Programmende, übertragen. Der CLR-Befehl gleicht alle weiteren Basic-Zeiger diesem Wert an.
In Kapitel 5 wird ein professionelles OLD vorgestellt, dessen Algorithmus diesem Einzeiler entnommen wurde; dennoch sollten Sie sich das OLD aus Kapitel 5 nicht entgehen lassen!
Noch etwas zum Listing: Im folgenden werden Sie oftmals Listings in doppelter Ausführung erhalten; einmal werden sie über LIST erstellt, das andere Mal mit der Routine »Übersichtli-

ches Listing« (Kapitel 5), welche jeden Doppelpunkt durch einen Zeilenvorschub (RETURN) ersetzt.
Die doppelte Darstellung bewährt sich außerordentlich bei der Analyse oder Änderung eines Programms, während die herkömmliche LIST-Ausgabe den Gesamtüberblick erleichtert.

Einzeiler #3: Reset ohne Datenverlust bei Fortsetzung des aktuellen Programms

Mit dem Begriff »Reset« assoziiert man auch den Abbruch eines laufenden Programms. Doch folgender Einzeiler führt lediglich einen Teil-Reset durch, der jedoch das laufende Basic-Programm nicht unterbricht:

Listing 2.3: EZ INITIALIZE

```
1 POKE648.4:SYS64789:SYS58451:SYS58784
2 :
3 REM *** RESET OHNE DATENVERLUST ***
4 REM ***   BEI FORTSETZUNG DES   ***
5 REM ***   AKTUELLEN PROGRAMMS   ***
6 :
7 REM   (C) 1985 BY FLORIAN MUELLER

READY.


1 POKE648.4
  SYS64789
  SYS58451
  SYS58784
2

3 REM *** RESET OHNE DATENVERLUST ***
4 REM ***   BEI FORTSETZUNG DES   ***
5 REM ***   AKTUELLEN PROGRAMMS   ***
6

7 REM   (C) 1985 BY FLORIAN MUELLER
```

Die detaillierte Wirkung der einzelnen SYS-Befehle entnehmen Sie am besten einer Systemdokumentation (siehe »C64 für Insider«).

Hier einige Anwendungsvorschläge:

- Unerwünschte Erweiterungen des Betriebssystems (Turbo-Tape) oder Basic-Interpreters (Basic-Expansion) werden fast ausnahmslos durch diese Zeile desaktiviert. Damit lassen sich auch bestimmt Kompatibilitätsprobleme bewältigen: So lassen sich manche Programme zwar mit Turbo-Tape laden, stürzen aber nach dem Start ab (Listing "Springvogel" aus der Ausgabe 11/84 des 64'er-Magazins); dieser Einzeiler schafft alle Schwierigkeiten aus der Welt, wenn er nach dem Laden, aber vor dem Starten des Programms eingegeben wird.

- Sie wollen alle Toolkits abschalten und den regulären Zeichensatz darstellen.
- Ein Programm mit hochauflösender Grafik, verändertem Zeichensatz und/oder Schutz-POKEs gegen LIST, RUN/STOP, SAVE ist beendet worden (Fehlermeldung, Abbruch, Programmende). Nach der Einzeiler-Eingabe (dann ohne Zeilennummer, im Direktmodus) stehen alle Funktionen wieder zur Verfügung.
- Wenn Sie Eingaben nach Abbruch eines Programms machen, erscheinen diese nicht auf dem Bildschirm, nur die Farbe ändert sich. In einem solchen Fall geben Sie den Einzeiler »blind« und ohne Zeilennummer ein.

Einzeiler #4: RENUMBER in Basic

Folgender Einzeiler will sich zwar nicht mit einem professionellen RENUMBER-Programm messen, aber er funktioniert:

Listing 2.4: EZ RENUMBER

```
1 FORA=2049TOPEEK(45)+PEEK(46)*256-3:POKEA+2,Z:POKEA+3,0:A=PEEK(A)+PEEK(A+1)*256
-1:Z=Z+1:NEXT
10 :
20 REM RENUMBER (GEORG WICHERT)

READY.

1 FORA=2049TOPEEK(45)+PEEK(46)*256-3
  POKEA+2,Z
  POKEA+3,0
  A=PEEK(A)+PEEK(A+1)*256-1
  Z=Z+1
  NEXT
10

20 REM RENUMBER (GEORG WICHERT)
```

Es werden die Zeilennummern eines Basic-Programms, das nicht länger als 255 Zeilen sein darf, in nur wenigen Sekunden neu numeriert (erste Zeilennummer = 0, Schrittweite = 1). Die Sprungadressen von GOTO/GOSUB/RUN bleiben jedoch unverändert und müssen von Hand ersetzt werden.

Beim Abtippen des Einzeilers ist zu beachten, daß die Befehle abgekürzt eingegeben werden müssen (siehe C64-Handbuch, Anhang D).

Nun zur Erklärung des Einzeilers: Um die Zeilen eines Programms umnumerieren zu können, muß man zunächst einmal wissen, an welchen Speicherplätzen es sich befindet. Jedes Basic-Programm belegt im Normalzustand des C64 die Speicherstellen 2048 bis PEEK(45)+256*PEEK(46)-3. Dazu gibt PEEK(2049)+256*PEEK(2050) an, bei welcher Adresse die erste Zeile aufhört. Die Adressen 2051 und 2052 geben Aufschluß über die erste Zeilennummer, die durch den Term PEEK(2051)+256*PEEK(2052) bestimmt wird. Die

zweite Zeilennummer findet man im Speicher an den Adressen PEEK(2049)+ PEEK(2050)*256+2 und PEEK(2049)+PEEK(2050)*256+3.
Zurück zum Renumber-Programm: Die Zeilennummern befinden sich jeweils an den Adressen A+2 (Low-Byte) und A+3 (High-Byte). Aus Platzgründen poken wir an die Stelle A+2 den Wert z (z=0,1,2...,n-1; n steht für die Anzahl der Zeilen des Basic-Programms, das umnumeriert werden soll, den Einzeiler eingeschlossen) und an die Stelle A+3 den Wert 0. Daher darf das Basic-Programm 255 Zeilen (maximale Grenze für 1 Byte) nicht überschreiten, denn dann müßten wir den Einzeiler wie folgt zum Zweizeiler abändern:

```
1 FOR A=2049 TO PEEK(45)+PEEK(46)*256-3:POKE A+3,z/256:POKE A+2,
  z-INT(z/256)*256
2 A=PEEK(A)+PEEK(A+1)*256-1:z=z+1:NEXT
```

Einzeiler #5: TRACE – die Kamera für die Programmablauf-Verfolgung

Beim Austesten eines Basic-Programms fehlt in Basic 2.0 die so wichtige Funktion des TRACE-Kommandos anderer Basic-Dialekte. Mit TRACE ist die schrittweise Abarbeitung eines Programms gemeint, welche die Fehlersuche erheblich vereinfacht. Sehr professionelle TRACE-Routinen geben nach jedem Befehl sogar die Zeilennummer sowie auf Wunsch das dazugehörige Listing und die Variableninhalte aus.
Folgendes Programm bildet ein Einzelschritt-Utility, bei dessen Ausführung nach jedem Befehl – auch im Direktmodus – die Taste <SHIFT> oder <C=> zu drücken ist:

Listing 2.5: EZ TRACE

```
1 POKE648,192:PRINT"(CLR)(RVS))(OFF)K(RVS)M(OFF)HC(RVS))-M(OFF)IC  (RVS) _(OFF)L
(RVS)_'":POKE648,4:SYS49152

READY.

1 POKE648,192
  PRINT"(CLR)(RVS))(OFF)K(RVS)M(OFF)HC(RVS))-M(OFF)IC  (RVS) _(OFF)L(RVS)_'"
  POKE648,4
  SYS49152
```

Aktiviert wird der Einzelschrittmodus durch RUN, abgeschaltet durch SYS 58451, und nach dem Ausschalten wieder mit SYS 49152 reaktiviert.
Jeder Basic-Befehl, auch LIST oder RUN, wird erst nach Drücken von <C=> oder <SHIFT> ausgeführt. Verantwortlich ist dafür ein kleines Maschinenprogramm, welches nach Verschiebung des Bildschirmspeichers (POKE 648,192) nach $c000 (neue Bildschirmspeicher-Basisadresse) geschrieben wird; anschließend schaltet das Basic-Programm den alten Bildschirmbereich ein (POKE 648,4), welcher bei 1024 ($0400) beginnt. Die im Maschinenprogramm vollzogene Vektorumbiegung stellt das Kernstück der Routine

dar; alle Einzelheiten, auch die geschickt realisierte Warteschleife, erklären sich am besten mit Hilfe einer Systemdokumentation (siehe »C 64 für Insider«).

Einzeiler #6: Directory betrachten

Hat man nun ein wichtiges Programm in Basic geschrieben und will die Früchte seiner Arbeit mit dem Abspeichern ernten, so kann es vorkommen, daß man die richtige Diskette nicht wiedererkennt. Wurde nicht zuvor ein Hilfsprogramm mit Directory-Anzeige (etwa Paradoxon-Basic, Kapitel 5) geladen, so hilft folgender Einzeiler weiter:

Listing 2.6: EZ DIRECTORY

```
0 GET#1,A$:A=ASC(A$+"(HOME)"):PRINTCHR$((A=130AND13OR((31<AANDA<95)ANDA)));:GOTO
0

READY.


0 GET#1,A$
  A=ASC(A$+"(HOME)")
  PRINTCHR$((A=130AND13OR((31<AANDA<95)ANDA)));
  GOTO0
```

Mit LOAD "$",8 und LIST würde aber das eigene Programm gelöscht und die gesamte Arbeit wäre umsonst. Mit dem beschriebenen Einzeiler und folgender Eingabe hingegen wird das Inhaltsverzeichnis der Diskette angezeigt:

```
OPEN 1,8,2,"$":GOTO 0
```

Auch sequentielle Textdateien können durch Abänderung des OPEN-Befehls dadurch ausgegeben werden. Sie erhalten jedoch noch wesentlich bessere Directory- und File-Dump-Routinen in den folgenden Abschnitten; die beschriebene Lösung ist aber ein interessanter Einzeiler zu diesem Thema.

Einzeiler #7: Speicherbereich sichern

Folgende Befehlsfolge speichert einen beliebigen Speicherbereich (Maschinenprogramm, Grafik-Bitmap o.ä.) auf ein Peripherigerät (Diskette, Datasette), ohne (!) die Basic-Zeiger zu verstellen:

```
SYS(57812)"BEISPIEL",8:POKE193,32:POKE194,78:POKE174,241:POKE175,85:
SYS62957
```

Daher kann dieses Verfahren auch innerhalb von Programmen eingesetzt werden. Für die Berechnung von LE/HE muß allerdings die Endadresse plus 1 genommen werden, weil die Endadresse für SAVE-Vorgänge die erste nicht mehr zu speichernde Adresse angibt. Als Beispiel wollen wir ein Programm im Bereich 20000–22000 unter den Namen »BEISPIEL« auf Diskette sichern:

Listing 2.7: EZ BEREICH-SAVE

```
1 SYS(57812)A$,X:POKE193,LS:POKE194,HS:POKE174,LE:POKE175,HE:SYS62957
2 :
3 REM MASCHINENPROGRAMM IM BEREICH VON
4 REM ADRESSE  [LS/HS]  BIS  [LE/HE]-1
5 REM UNTER DEM FILENAMEN [A$] AUF DAS
6 REM PERIPHERIEGERAET [X] ABSPEICHERN

READY.


1 SYS(57812)A$,X
  POKE193,LS
  POKE194,HS
  POKE174,LE
  POKE175,HE
  SYS62957
2

3 REM MASCHINENPROGRAMM IM BEREICH VON
4 REM ADRESSE  [LS/HS]  BIS  [LE/HE]-1
5 REM UNTER DEM FILENAMEN [A$] AUF DAS
6 REM PERIPHERIEGERAET [X] ABSPEICHERN
```

Falls der Einzeiler infolge eines längeren Filenamens zu lang für den Editor werden sollte, sind die POKE-Befehle einfach durch <P> <SHIFT O> abzukürzen.

Einzeiler #8: GOTO-X

Das Basic 2.0 läßt hinter GOTO nur absolute Zeilenangaben, also feststehende Zahlen, zu; Variablen werden jedoch nicht akzeptiert, obwohl dies in vielen Fällen die Programmierung von Sprungverteilern um einiges vereinfachen würde.
Folgendes Programm simuliert den neuen Befehl GOTO X:

Listing 2.8: EZ GOTO-X

```
10 SYSPEEK(61)+256*PEEK(62)+26:"▛L(F1)E(F1)F▏(LEFT)(F1)U(F1)V  ($12) ▏(UP)U▬(ORNG)
╲▏(F4)(UP)U▪($128)¬▩.▏ -  (DOWN)┴ +▁ ($1)▔L▔▩"
20 :
30 REM SYNTAX:
40 REM LL%=ZIELZEILE:GOTO 10
```

Zur Benutzung: Man weist der Variablen LL% den Wert der anzuspringenden Zeile zu und springt mit GOTO zur Zeile, in welcher GOTO X steht (im Beispiel: Zeile 10). Das seltsame Aussehen der Zeile läßt sich folgendermaßen erklären: In den Anführungszeichen steht ein Maschinenprogramm, dessen Opcodes im Listing des Basic-Interpreters einige merkwürdige

Zeichen erzeugen. Das Programm adressiert nur relativ, damit die Programmzeile frei verschieblich (relokatibel) bleibt.
Der mit der Materie etwas vertrautere Leser (siehe »C64 für Insider«) wird feststellen, daß das Programm recht umständlich geschrieben ist. Dies liegt daran, daß nicht alle CHR$-Codes, die einem Opcode entsprechen, eindeutig einer ASCII-Zahl zugeordnet sind. So gäbe zum Beispiel die Zahl 255 im Maschinenprogramm das Zeichen <C= K> ; beim Editieren aber würde das Basic dieses Zeichen als 161 auffassen. Dann wiederum ließe sich die Zeile nicht als Basic-Zeile editieren, was einer Veränderung der Zeilennummer als nahezu unüberwindliches Hindernis im Weg stehen würde. Aus diesem Grund wurden alle »doppeldeutigen« Codes umgangen, was die Umständlichkeit des Maschinenprogramms zur Folge hatte.
Die SYS-Anweisung stellt übrigens die Lage des Maschinenprogramms mit Hilfe zweier Basic-Zeiger fest und startet das Programm somit auch nach beliebig vielen Verschiebungen an der korrekten Einsprungadresse.

Einzeiler #9: Übersichtliche Formatierung am Bildschirm

Beim Editieren eines Basic-Programms eliminiert der Interpreter alle Leerzeichen zwischen der Zeilennummer und dem ersten Befehl (siehe »C64 für Insider«). Dies verhindert eine in anderen Programmiersprachen (C, Pascal) und Basic-Versionen durchaus gängige Formatierung durch führende Leerzeichen, weshalb viele Programmierer auf Doppelpunkte ausweichen. Es geht jedoch auch anders:

Listing 2.9: EZ SPACING

```
1  FORI=131TO128STEP-1:POKEI,234:NEXT:END
2              BEISPIEL: DIESE ZEILE IST
3                        WEIT EINGERUECKT
```

Dieses Programm modifiziert die im RAM stehende CHRGET-Routine, den wohl meistfrequentierten Teil des Basic-Interpreters, mit wenigen Handgriffen so, daß er alle eingegebenen Zeichen speichert.
Die Zeilen 2 und 3 im Beispiel sind der Beweis dafür. Der Nachteil liegt jedoch darin, daß bei aktivierter Routine eine Programmausführung nicht möglich ist (alle Leerzeichen führen zu SYNTAX ERROR). Deshalb müssen Sie nach Abspeichern des formatierten Programms den Computer in den Ausgangszustand versetzen (Reset); das zuvor eingegebene Programm behält jedoch, solange Sie es nicht mehr editieren, die SPACE-Einrückungen bei – und bleibt ablauffähig!

Einzeiler #10: LIST ohne Programmabbruch

Der gravierendste Nachteil des LIST-Befehls liegt darin, daß er innerhalb von Programmen praktisch nur an deren Ende verwendbar ist; nach LIST wird nämlich das Programm beendet.

Dahinter steht kein höherer Sinn, ganz im Gegenteil: Diese Einschränkung ist absolut überflüssig. Durch den simulierten Direktmodus gibt es zwar ebenfalls interessante Auswege, aber eine noch bessere Lösung demonstriert folgende Zeile:

Listing 2.10: EZ PROG.LIST

```
10 POKE768,61:SYS42980,LIST 10:POKE768,138:WAIT198,1:POKE198,0:GOTO10

10 POKE768,61
   SYS42980,LIST 10
   POKE768,138
   WAIT198,1
   POKE198,0
   GOTO10
```

Dieses Programm listet sich selbst immer wieder und wartet dabei permanent auf Tastendrücke von Ihrer Seite. Der Befehl LIST 10 wird hier nicht normal ausgeführt, sondern an die Interpreterschleife per SYS übergeben. Das Verbiegen des Vektors 768/769 bewirkt, daß die Interpreterschleife (siehe »C64 für Insider«) nach Ausführung von »LIST 10« ins Programm zurückspringt. Daher sollte das Programm nicht über <RUN/STOP> unterbrochen werden.

Listing 2.11: EZ HELP

```
1 POKE769,177:FORI=1TO76:POKE73,255:POKE781,I:SYS42794:PRINT,:NEXT:POKE769,227
2 :
3 REM *** HELP ***
4 :
5 REM ***   PROGRAMM DARF NICHT   ***
6 REN ***   UNTERBROCHEN WERDEN   ***

1 POKE769,177
  FORI=1TO76
  POKE73,255
  POKE781,I
  SYS42794
  PRINT,
  NEXT
  POKE769,227
2

3 REM *** HELP ***
4

5 REM ***   PROGRAMM DARF NICHT   ***
6 REN ***   UNTERBROCHEN WERDEN   ***
```

Einzeiler #11: Basic-Befehle aufgezählt

Vor allem als Einsteiger fällt es einem recht schwer, die zahlreichen Anweisungen des Basic 2.0 im Gedächtnis zu behalten. Dafür bietet sich die Routine von Seite 67 an, die alle verfügbaren Basic-Anweisungen aufzählt:
In manchen Basic-Erweiterungen wird dies als HELP- oder ORDER-Funktion bezeichnet. Vorsicht: Das Programm darf, wie Einzeiler #10, nicht unterbrochen werden; ansonsten gerät der C64 in eine Strudel von undefinierten Aktionen, und Sie müssen ihn ausschalten bzw. einen Reset auslösen. Die Ausgabeschleife stützt sich auf einen Teil der LIST-Routine, welcher die Basic-Tokens dekodiert (siehe »C64 für Insider«).

Einzeiler #12: DI-AS, ein grafischer Disassembler

Der Speicherbereich des C64 läßt sich in 256 Seiten (Pages) zu je 256 Byte unterteilen. Diese beginnen an einer Adresse mit dem Low-Byte 0 ($000, $0100, $0200 usw. bis $ff00). DI-AS interpretiert die Speicherinhalte als Bildschirmcodes und stellt die 64 Kbyte des C64 Seite für Seite auf den Bildschirmzeilen 7–13 dar. Die Seitennummer wird immer am oberen Bildschirmrand angezeigt.
Bevor das Programm geladen und gestartet wird, muß unbedingt mit SYS64738 ein Reset durchgeführt werden, da sonst möglicherweise die Zeiger der Zeropage nicht korrekt initialisiert werden. Es empfiehlt sich, zuvor ebenfalls den Bildschirm mit CLR zu löschen, um die Übersichtlichkeit zu erhöhen.
Dieses Programm wird mit RUN gestartet:

Listing 2.12: EZ DI-AS

```
5 PRINT"(HOME)(RVS)1(OFF)B(RVS)Q(OFF)((RVS)┌▄(OFF)  "PEEK(3)"(LEFT)  ":POKE41,5:G
ETA$:POKE3,PEEK(3)-(A$="(RGHT)")+(A$="(DOWN)"):SYS1024:GOTO5

5 PRINT"(HOME)(RVS)1(OFF)B(RVS)Q(OFF)((RVS)┌▄(OFF)  "PEEK(3)"(LEFT)  "
  POKE41,5
  GETA$
  POKE3,PEEK(3)-(A$="(RGHT)")+(A$="(DOWN)")
  SYS1024
  GOTO5
```

Dem Benutzer stehen nun folgende Optionen offen:

1. <CURSOR RIGHT> blättert im Speicher vorwärts.
2. <CURSOR DOWN> blättert rückwärts.
3. <RUN/STOP> POKE 3,nr:CLR:RUN <RETURN> betrachtet gezielt die Seite »nr« (0–255).

Fehlerbehandlung: Der Versuch, Seiten kleiner als 0 oder größer als 255 anzusehen, wird mit ILLEGAL QUANTITY ERROR quittiert. RUN bewirkt einen Neustart. Gehen Sie einige der folgenden Anwendungsbeispiele durch, um sich mit der Speicheraufteilung Ihres Computers vertraut zu machen:

Seite 0: In Zeile 5 ist die vom Betriebssystem versorgte Uhr. Sie sehen, wie von rechts nach links stückweise weitergezählt wird, da sich die entsprechenden Zeichen ändern.

Seite 1: Der Stapel der CPU, in welchem vor allem die untersten Zeilen regelmäßig geändert werden.

Seite 2: Oberste 2 Zeilen = Eingabepuffer. Hier stehen Ihre letzten Eingaben.

Seiten 4–7: Der Bildschirmspeicher selbst bildet sich ab. Halten Sie das Programm an, schreiben Sie ein paar Sätze in die untere Bildschirmhälfte und sehen Sie sich an, was sich verändert hat. Außerdem sind die Sprite-Zeiger zu sehen.

Seite 8: Hier steht unser Basic-Programm. Erkennen Sie es wieder?

Seite 160: Das CBM-Basic-ROM.

Seite 161: Die Liste der Basic-Befehle.

Seite 162: Die Fehlermeldungen.

Seite 208: Der VIC. Durch hektisches Flackern fällt das Kontrollregister $d011 sowie das Raster-Register $d012 auf.

Seiten 220–223: Die CIA-Register. Beachten Sie die Timer.

Seiten 228–255: Das Betriebssystem mit einigen Systemmeldungen.

Anmerkung: Manches läßt sich erst entdecken, wenn durch gleichzeitiges Drücken von <SHIFT C=> der Zeichensatz geändert wird. Nun nehmen Sie doch einmal alle möglichen Programme und schauen, wo Text, Befehle und sonstige Daten stehen!
Die Funktionsweise des Programms besteht darin, daß Adresse 3 die aktuelle Seitennummer enthält, wogegen Adresse 2 immer auf 0 steht. Dadurch kann die Maschinenroutine in einer indizierten Schleife die Inhalte aus der aktuellen Seite über

```
LDA (2),Y
```

einlesen und über

```
STA xxxx,Y
```

direkt in den Bildschirmspeicher übernehmen.

Einzeiler #13: Das kürzeste Heimcomputer-Orgelprogramm

Nach dem Start folgenden Programms entsteht im Bildschirmspeicher ein Maschinenprogramm von 45 Byte Länge:

Listing 2.13: EZ MICROSOUND

```
1 PRINT"(CLR)(RVS))OM(OFF)X(RVS)|)┌M(OFF)E(RVS)|┐(OFF) (RVS)┐ (OFF)P(RVS)L(OFF)D
(RVS)|╱(OFF) (RVS)┐%╯┌(OFF)S)?(RVS)M(OFF)A(RVS)| |L(OFF)D(RVS)|-(OFF)R(RVS)┐╲(OFF
)R(RVS)┐┬╳┌▌(OFF) ":SYS4↑5
```

Es wird gleichzeitig mit SYS1024 gestartet und ist meines Wissens das kürzeste Heimcomputer-Orgelprogramm der Welt. 86 Basic-Bytes erzeugen nur 45 Byte Maschinencode, wovon 14 Byte für Grafik-Effekte und 3 Byte für die Programmende-Abfrage aufgewandt wurden.
Das Musik-Programm erzeugt zu jeder ausgelösten Taste einen eigenen Ton und wird mit <INST/DEL> beendet. Es funktioniert im wesentlichen durch Verknüpfung des Codes für die gedrückte Taste (siehe »C 64 für Insider«) mit 63 und dem Setzen als High-Byte für Stimme 1 des SID; der Grafik-Effekt kommt durch Vergleich mit dem Rasterregister (Adresse 53266) zustande. Alle Tasten außer <CTRL>, <C=>, <SHIFT> und <RESTORE> sind dabei mit Noten belegt, die überwiegend so angeordnet sind, daß die weiter links stehenden Tasten tiefere Töne spielen als die rechten – also wie eine Elektro-Orgel! Die Funktions- und Cursor-Tasten ergeben absolute Bässe.
Maschinensprache-Freaks können sich selbst ein Monitor-Listing erstellen.

Einzeiler #14: Speicherblockverschiebung blitzschnell

Dieses Programm dient zur Übertragung von Speicherblöcken:

Listing 2.14: EZ BLOCKMOVE

```
1 POKE95,0:POKE96,160:POKE90,0:POKE91,192:POKE88,0:POKE89,192:SYS41919
2 :
3 REM ↑ BASIC-ROM INS RAM KOPIEREN
5 :
6 :
10 REM *** PARAMETER: ***
11 POKE95,AL:POKE96,AH:POKE90,EL:POKE91,EH:POKE88,NL:POKE89,NH:SYS41919


1 POKE95,0
  POKE96,160
  POKE90,0
  POKE91,192
  POKE88,0
  POKE89,192
  SYS41919
2
```

```
3 REM ↑ BASIC-ROM INS RAM KOPIEREN
5

6

10 REM *** PARAMETER
   ***
11 POKE95,AL
   POKE96,AH
   POKE90,EL
   POKE91,EH
   POKE88,NL
   POKE89,NH
   SYS41919
```

Die Variablen mit L sind dabei jeweils das Low-, diejenigen mit H im Variablennamen das High-Byte einer Adresse:

AL/AH: exakte Anfangsadresse des Quellbereiches
EL/EH: Endadresse des Quellbereiches +1
NL/NH: Endadresse des Zielbereiches +1

Der Einzeiler stützt sich auf die BLTUC-Routine im C64-ROM (siehe »C64 für Insider«). Die Beispielzeile kopiert den Basic-Interpreter ins RAM und sollte von Ihnen für einige der nun folgenden Tricks im Auge behalten werden.

Einzeiler #15: AND und OR für gesamten Zahlenbereich

Bei AND/OR ist nur ein stark eingeschränkter Zahlenbereich zulässig. Folgendes Listing schafft Abhilfe, ist aber eigentlich ein Zweizeiler:

Listing 2.15: EZ AND/OR

```
1 POKE95,0:POKE96,160:POKE90,0:POKE91,192:POKE88,0:POKE89,192:SYS41919
2 H=45505:POKEH,41:POKEH+1,0:POKEH+2,240:POKE1,54

1 POKE95,0
  POKE96,160
  POKE90,0
  POKE91,192
  POKE88,0
  POKE89,192
  SYS41919
2 H=45505
  POKEH,41
  POKEH+1,0
  POKEH+2,240
  POKE1,54
```

Die eigentliche AND/OR-Modifikation steht erst in Zeile 2 (also doch ein Einzeiler!), vorausgeschickt wird lediglich die BLOCKMOVE-Variante von Einzeiler #14. Danach ist AND/OR für den gesamten Integerbereich (0 bis 65535) verwendbar.
Das Funktionsprinzip der ROM-Modifikation ist mit einem Monitor sowie einem ROM-Listing und einer Systemdokumentation (siehe »C64 für Insider«) für Maschinensprache-Interessierte sehr transparent.

Einzeiler #16: Verbesserte Eingaberoutine als INPUT-Ersatz

Die Idee: Eine INPUT-Routine, die den normalen INPUT-Befehl vollständig zu ersetzen vermag, aber zusätzlich Satzzeichen wie Komma, Doppelpunkt und Strichpunkt als Eingabe erlaubt. Alle sonstigen Vor- und Nachteile bleiben erhalten.

Die Wirkung: Alle Zeichen der Tastatur werden übernommen, auch Leerzeichen vor beginnendem Text (führende Leerzeichen).

Das Programm:

Listing 2.16: EZ INPUT

```
1 SYS42336:XX$="":FORII=512TO600:AA=PEEK(II):IFAATHENXX$=XX$+CHR$(AA):NEXT
10 :
20 REM INPUT


1 SYS42336
  XX$=""
  FORII=512TO600
  AA=PEEK(II)
  IFAATHENXX$=XX$+CHR$(AA)
  NEXT
10

20 REM INPUT
```

Die zentrale Funktion im Programm kommt der Eingaberoutine ab Adresse 42336 (siehe »C64 für Insider«) zu. Diese schreibt alle, von einer Bildschirmzeile (80 Zeichen pro logische Zeile) erfaßten Zeichen in den Basic-Eingabepuffer, welcher bei Adresse 512 beginnt und durch ein Nullbyte abgeschlossen wird. Der Rest des Einzeiler-Programms liest nun zeichenweise bis zur genannten 0 den Eingabepuffer aus, und stellt dabei den String XX$ zusammen. Die Schleife ist zwar auf 88 Durchläufe programmiert, wird aber niemals so weit kommen, da die Zeichenzahl durch den Bildschirm begrenzt ist (2*40 Zeichen/Zeile = 80 Zeichen pro logische Zeile). Erfolgt keine Eingabe (es wird nur die RETURN-Taste betätigt), so wird das Programm mit XX$=CHR$(32) verlassen. Ansonsten enthält XX$ alle sichtbaren, eingegebenen Zeichen (also alle Zeichen mit Ausnahme der Steuerzeichen).

Die Variablen: AA enthält den aktuellen ASCII-Code der Eingabe, II ist die Schleifenvariable (Adresse in Systemeingabepuffer) und in XX$ befindet sich der bislang ausgewertete, eingegebene Text.
Es sei darauf hingewiesen, daß Sie an anderer Stelle in diesem Buch eine wesentlich professionellere Eingaberoutine erhalten. Vom Funktionsprinzip her ist der hier vorgestellte Einzeiler jedoch hochinteressant, und außerdem macht ihn seine Kürze und unkomplizierte Bedienung zu einem wertvollen Modul für jegliche Weiterverwendung.

Einzeiler #17: Window-Technik

Der Begriff »Window-Technik« ist eine hochgestochene Bezeichnung für das, was folgender Einzeiler leistet:

Listing 2.17: EZ PSEUDO-WINDOW

```
1 FORI=40960TO49151:POKEI,PEEK(I):NEXT:FORI=57344TO65535:POKEI,PEEK(I):NEXT:POKE
59639,10:POKE1,53
2 :
3 REM ↑ AUSFUEHRUNGSZEIT: ETWA 71 SEK.
4 :
5 REM *** ABSCHALTBAR UEBER
6 REM *** <RUN/STOP>+<RESTORE>

1 FORI=40960TO49151
  POKEI,PEEK(I)
  NEXT
  FORI=57344TO65535
  POKEI,PEEK(I)
  NEXT
  POKE59639,10
  POKE1,53
2

3 REM ↑ AUSFUEHRUNGSZEIT
   ETWA 71 SEK.
4

5 REM *** ABSCHALTBAR UEBER
6 REM *** <RUN/STOP>+<RESTORE>
```

Dabei werden die obersten 10 Zeilen nicht mehr beim Scrolling (Abrollen des Bildschirms nach oben) beeinflußt. Der Vorteil: Dort lassen sich Informationen von bleibender Wichtigkeit festhalten. Kopfzeilen und Tabellen werden also von diesem Programm unterstützt.
Es benötigt zunächst etwa 71 Sekunden, um das Betriebssystem, genauer gesagt: dessen SCROLL-Routine (siehe »C64 für Insider«), unseren Wünschen anzupassen. Danach kann jederzeit durch POKE 59639,x die Anzahl derjenigen Zeilen, die gegen Scrolling immun sind, verändert werden.

Einzeiler #18: Smooth-Scrolling

Einer der optisch ansprechendsten Grafikeffekte besteht im sanften Abrollen am Bildschirms (Soft-Scrolling oder Smooth-Scrolling genannt). Hier ist ein Beispielprogramm, daß ein Scrolling in X-Richtung (horizontal) realisiert:

Listing 2.18: EZ SMOOTH-SCROLL

```
10 FORT=1TO7:POKE53270,T:FORG=0TO3:NEXT:NEXT:ONAGOTO10:FORY=1024TO2023:POKEY,194
:NEXT:A=1:GOTO10
20 :
30 REM *** SCROLLEN IN X-RICHTUNG ***
40 REM
50 REM BEMERKENSWERTER PROGRAMMIERTRICK:
60 REM [ON ... GOTO] ALS ERSATZ FUER
70 REM                IF-ABFRAGE


10 FORT=1TO7
   POKE53270,T
   FORG=0TO3
   NEXT
   NEXT
   ONAGOTO10
   FORY=1024TO2023
   POKEY,194
   NEXT
   A=1
   GOTO10
20

30 REM *** SCROLLEN IN X-RICHTUNG ***
40 REM
50 REM BEMERKENSWERTER PROGRAMMIERTRICK

60 REM [ON ... GOTO] ALS ERSATZ FUER
70 REM                IF-ABFRAGE
```

Die Verzögerung wird von der Schleife FORG=0TO3:NEXT bewirkt und ist somit veränderlich. Ein bemerkenswerter Programmiertrick besteht im ON-GOTO-Befehl, welcher hier einen sinnvollen Ersatz einer IF-Abfrage zuläßt. Ist A nämlich < > 1, so wird der Programmbereich nach ON A GOTO 10 ausgeführt.

Einzeiler #19: Text-Scrolling

Dieser Einzeiler stellt den Inhalt der Variablen A$ am Bildschirm dar und läßt ihn zeichenweise nach links abrollen (siehe Seite 75).
Die Variable A$ ist vor Ausführung des Einzeilers als Parameter zu definieren sowie die Adresse A mit 53270 (Register für horizontales Scrolling) und L mit der Anzahl der Zeichen, die gleichzeitig am Bildschirm erscheinen sollen, also normalerweise mit 40, zu belegen. Beispiel:

```
L=40:A=53270:A$="*** SCROLL SOFTLY AND SMOOTHLY ***":GOTO 1
```

Listing 2.19: EZ SCROLL-X

```
1 FORR=1TOLEN(A$):FORI=207 TO200STEP-1:PRINT"(HOME)"MID$(A$,R,L):POKEA,I:NEXTI,R
10 :
20 REM SCROLL


1 FORR=1TOLEN(A$)
  FORI=207 TO200STEP-1
  PRINT"(HOME)"MID$(A$,R,L)
  POKEA,I
  NEXTI,R
10

20 REM SCROLL
```

Das Prinzip: Der Text wird auf den Bildschirm ausgegeben sowie der Bildschirminhalt mit Hilfe des Smooth-Scrolling-Registers A=53270 punktweise nach links gezogen, bis er um 7 Punkteinheiten verschoben worden ist. Nun wird der gesamte Textbereich nach links geschoben und fast gleichzeitig das Scroll-Register zurückgesetzt, so daß es aussieht, als sei der Text um den achten Punkt verschoben worden. Hinweise:

- Vor der Benutzung empfiehlt es sich sehr, den Bildschirm zu löschen da auch der restliche Bildschirminhalt verschoben würde.
- Außer dem verwendeten <HOME> können auch andere Cursor-Steuerzeichen zur Positionierung des Textes verwendet werden.

Einzeiler #20: Abwärts-Scrolling

Aller guten Dinge sind drei: Nach zwei verschiedenen Einzeilern zum horizontalen Scrolling schließt nun einer zum Abwärts-Scrollen die Serie ab.

Listing 2.20: EZ DOWN-SCROLL

```
1 PRINT"(HOME)(DOWN)(LEFT)(INST)":POKE218,152
```

Diese bewirkt durch die Steuerzeichenfolge

<HOME> <DOWN> <CURSOR LEFT> <INSERT>

ein Abwärts-Scrolling des gesamten Bildschirms bis auf die oberste Zeile, wozu auch die Adresse 218 auf 152 gesetzt wird (siehe »C64 für Insider«).

Einzeiler #21: Zeile löschen

Dieser kleine Einzeiler löscht bestimmte Zeilen auf dem Bildschirm:

Listing 2.21: EZ DELETE-ZEILE

```
10 FORLN=VTOB:POKE781,LN:SYS59903:NEXT
20 :
30 REM FUER NUR EINE ZU LOESCHENDE ZEILEGILT:
40 :
50 POKE780,LN:SYS59903
60 :
70 REM DELETE-ZEILE   STEFAN KEIMEIER


10 FORLN=VTOB
   POKE781,LN
   SYS59903
   NEXT
20

30 REM FUER NUR EINE ZU LOESCHENDE ZEILEGILT

40

50 POKE780,LN
   SYS59903
60

70 REM DELETE-ZEILE   STEFAN KEIMEIER
```

Dabei wird eine ROM-Routine des C64 aufgerufen (siehe »C64 für Insider«), die eine Zeile vom Bildschirm löscht, deren Nummer (0–24) im X-Register steht. Zunächst wird die Zeilennummer in die Speicherzelle geschrieben, deren Inhalt der SYS-Befehl ins X-Register übernimmt. Hierbei bleibt die Cursorposition unbeeinflußt. Diese Übergabevariablen werden benötigt:

V = Anfangszeile (Von Zeile)
B = Endzeile (Bis Zeile)
Die Schleifenvariable heißt LN.

Einzeiler #22: Zeilen kopieren

Bleiben wir bei der Bearbeitung von Bildschirmzeilen. Folgender Einzeiler auf Seite 77 bewirkt ein Scrolling, das insgesamt so aussieht, als würde der Bildschirm rotieren.
Die Routine eignet sich aufgrund ihrer außergewöhnlichen Geschwindigkeit in besonderem Maße für Spiele. So wurde mit ihrer Hilfe in nur 23 Basic-Zeilen ein grafisches Frogger-Spiel erstellt (siehe 2.2).

Listing 2.22: EZ ZEILEN-COPY

```
1 PRINT"(UP)      ":FORT=0TO1:POKE173,PEEK(60656):POKE780,PEEK(216):SYS59848:PRINT
:WAIT203,64:T=0:NEXT


1 PRINT"(UP)      "
  FORT=0TO1
  POKE173,PEEK(60656)
  POKE780,PEEK(216)
  SYS59848
  PRINT
  WAIT203,64
  T=0
  NEXT
```

Einzeiler #23: Rechtsbündige Zahlenausgabe

So gibt man den Inhalt der Variablen A rechtsbündig aus:

Listing 2.23: EZ PRINT-USING

```
20 B=ABS(A):PRINTTAB(INT(LOG(B-(B=0))*.43429448188)*(B>=1)+INT(-B)*(B<1)+22);A
30 :
40 REM *** GIBT DIE ZAHL [A]
50 REM *** RECHTSBUENDIG AUS


20 B=ABS(A)
   PRINTTAB(INT(LOG(B-(B=0))*.43429448188)*(B>=1)+INT(-B)*(B<1)+22);A
30

40 REM *** GIBT DIE ZAHL [A]
50 REM *** RECHTSBUENDIG AUS
```

Durch die Verwendung eines Zehnerlogarithmus mittels Multiplikation mit der Konstante 0.43429448188 kann die Anzahl der Stellen ermittelt werden. Zudem wird noch das Vorzeichen berücksichtigt.

Einzeiler #24: Zahlenformatierung

Enthält X eine auszugebende Zahl, A die Anzahl der gewünschten Vor- und B die der Nachkommastellen, so übernimmt dieser Einzeiler die entsprechende Formatierung, ohne zu runden (siehe Seite 78):

Listing 2.24: EZ ZAHLFORMAT

```
10 PRINT TAB(A-(INT(LOG(X)/LOG(10))));INT(X*10↑B)/10↑B
20 :
30 REM [A] - ANZAHL DER VORKOMMASTELLEN
40 REM [B] - ANZAHL DER NACHKOMMASTELLEN
50 REM [X] - AUSZUGEBENDE ZAHL
```

Beispiel:

```
X=57.503:A=3:B=2:GOTO 10
```

Einzeiler #25: Text-Zentrierung

Soll ein bestimmter Text, enthalten in A$, schön in der Mitte einer Zeile stehen (Zentrierung), so bedient man sich einfach folgender Anweisungen:

```
A$="TEXT":GOTO 1
```

Listing 2.25: EZ ZENTRIERUNG

```
1 FOR I=1 TO (40-LEN(A$))/2:PRINT" ";:NEXT:PRINTA$
2 :
3 REM ZENTRIERUNG VON [A$]


1 FOR I=1 TO (40-LEN(A$))/2
  PRINT" ";
  NEXT
  PRINTA$
2

3 REM ZENTRIERUNG VON [A$]
```

Es werden solange Leerzeichen ausgegeben, bis das Drucken des eigentlichen Textes exakt an der Position erfolgt, durch welche der Text schließlich in die Mitte zu stehen kommt.

Einzeiler #26: Text-Rechtsbündigkeit

Soll ein Text soweit eingerückt werden, daß das Textende mit dem Zeilenende zusammenfällt (was man als rechtsbündigen Druck bezeichnet), so bietet sich folgende Anweisung an:

```
A$="TEXT":GOTO 1
```

Listing 2.26: EZ RECHTSBNDG

```
1 FOR I=1 TO 40-LEN(A$):PRINT" ";:NEXT:PRINTA$
2 :
3 REM RECHTSBUENDIGE AUSGABE VON [A$]
```

```
1 FOR I=1 TO 40-LEN(A$)
  PRINT" ";
  NEXT
  PRINTA$
2

3 REM RECHTSBUENDIGE AUSGABE VON [A$]
```

Das Funktionsprinzip ist mit Einzeiler #25 eng verwandt, allerdings werden zum rechtsbündigen Druck in der Regel mehr Leerzeichen zur Einrückung benötigt.

Einzeiler #27: Zahlenkonvertierung »Dezimal → Beliebiges Zahlensystem«

Beim Computern kommt man oftmals nicht drum herum, sich verschiedener Zahlensysteme zu bedienen:

- Das herkömmliche Dezimalsystem ist genau das Zahlensystem, mit dem man täglich arbeitet.
- Der Computer selbst operiert meist im Binärsystem (auch Dualsystem oder Zweiersystem genannt).
- Einen sinnvollen Kompromiß stellt das Hexadezimalsystem (Sedezimalsystem = 16er-System) dar (angeblich deshalb, weil es weder die Menschen noch die Computer verstehen...)

Folgender Einzeiler stellt die Zahl D (Dezimalzahl) im Zahlensystem der Basis B (Beispiele: 2 = Binärsystem, 16 = Hexadezimalsystem) im String Z$ dar:

Listing 2.27: EZ DEZIMAL → BEL.

```
10 Z$="":FORP=0TO0:D=D/B:S=(D-INT(D))*B:Z$=CHR$(55+S+7*(S<10))+Z$:P=-D:NEXT
20 :
30 REM *** DEZIMAL->BELIEBIG ***
40 REM
50 REM [D] = DEZIMALER ZAHLENWERT
60 REM [B] = GEWUENSCHTES ZAHLENSYSTEM
70 REM       (BASIS)
80 REM
90 REM ERGEBNIS IN [Z$]
```

```
10 Z$=""
   FORP=0TO0
   D=D/B
   S=(D-INT(D))*B
   Z$=CHR$(55+S+7*(S<10))+Z$
   P=-D
   NEXT
20

30 REM *** DEZIMAL->BELIEBIG ***
40 REM
50 REM [D] = DEZIMALER ZAHLENWERT
60 REM [B] = GEWUENSCHTES ZAHLENSYSTEM
70 REM       (BASIS)
80 REM
90 REM ERGEBNIS IN [Z$]
```

Beispiel:

```
D=64738:B=16:GOTO 10
```

liefert in Z$ das Ergebnis »FCE2«.

Einzeiler #28: Zahlenkonvertierung »Beliebiges Zahlensystem → Dezimal«

Dies ist die umgekehrte Operation von Einzeiler #27. Diesmal gibt man in Z$ den Zahlenstring im beliebigen Zahlensystem an, nennt in B die Basis der Z$-Darstellung und erhält prompt in D den dezimalen Wert.

Listing 2.28: EZ BEL. → DEZIMAL

```
10 D=0:FORS=1TOLEN(Z$):H=ASC(MID$(Z$,S))-48:D=D*B+H+7*(H>9):NEXT
20 :
30 REM *** BELIEBIG->DEZIMAL ***
40 REM
50 REM [Z$] = ZAHLENWERT IM SYSTEM MIT
60 REM [B]  = BASIS DES ZAHLENSYSTEMS
70 REM
80 REM ERGEBNIS IN [D]
```

```
10 D=0
   FORS=1TOLEN(Z$)
   H=ASC(MID$(Z$,S))-48
   D=D*B+H+7*(H>9)
   NEXT
20

30 REM *** BELIEBIG->DEZIMAL ***
40 REM
50 REM [Z$] = ZAHLENWERT IM SYSTEM MIT
60 REM [B]  = BASIS DES ZAHLENSYSTEMS
70 REM
80 REM ERGEBNIS IN [D]
```

Beispiel:

```
Z$="FCE2":B=16:GOTO 10
```

liefert in D das Ergebnis 64738.

Einzeiler #29: Division mit beliebig vielen Nachkommastellen

Folgender Einzeiler stellt die Lösung eines wichtigen mathematischen Problems dar: Wie dividiert man eine Zahl Z durch eine andere Zahl D bei Berücksichtigung von N Nachkommastellen?

Listing 2.29: EZ DIVISION

```
0 INPUTZ,D,N:Q%=Z/D:PRINTQ%:R=Z-D*Q%:FORI=1TON:E%=R*10/D:PRINT"(LEFT)";E%;:R=10*
R-D*E%:NEXT
1 REM
2 REM *** DIVISION  MIT  BELIEBIG ***
3 REM *** VIELEN NACHKOMMASTELLEN ***


0 INPUTZ,D,N
  Q%=Z/D
  PRINTQ%
  R=Z-D*Q%
  FORI=1TON
  E%=R*10/D
  PRINT"(LEFT)";E%;
  R=10*R-D*E%
  NEXT
1 REM
2 REM *** DIVISION  MIT  BELIEBIG ***
3 REM *** VIELEN NACHKOMMASTELLEN ***
```

Zunächst erfolgt die Eingabe von Z, D und N über INPUT. Anschließend wird im Programmlauf jede Stelle ausgegeben; die letzte gibt dann die gesuchte »N-te« Kommastelle an.

Variablenliste:

Z = Zahl (Dividend)
D = Divisor
N = Anzahl der gewünschten Stellen hinter dem Komma
E% = Ergebnis ohne Rest (wird ausgegeben)
R = Rest
Q% = Quotient, beschränkt auf Vorkommazahl

Beschreibung:

Um die Vorkommazahl zu erhalten, benötigt man Q=INT(Z/D), was ich durch Q%=Z/D vereinfacht habe. Q% wird als Vorkommazahl ausgegeben. Nun wird der Rest, nämlich die Diffe-

renz aus Z und Q%*D, gebildet. Dieser Rest dient im folgenden als Ausgangspunkt für die folgende Berechnung. Die Schleife wird N-mal durchlaufen, um exakt N Stellen hinter dem Komma zu ermitteln.
Analog zum Vorschleifenteil wird E%=R*10/D gebildet und ausgegeben als »I-te« Stelle (I=Schleifenvariable). Der neue Rest wird durch R=10*R-D*E% (Vergleich des Vorkommateils) berechnet. Nun wird die Schleife wiederum durchlaufen, bis alle Stellen ausgegeben sind. Die Verwendung von Integervariablen dient zum Einsparen längerer INT-Anweisungen.

Beispiel für den Programmablauf: Z=116, D=13, N=85; nach dem Programmlauf ergibt sich als letzte, also 85ste Stelle die 9.

Einzeiler #30: Fakultät

Bei der Besprechung sogenannter strukturierter Programmiersprachen (C, Pascal, TrueBasic) ist neben dem berühmten »Sieve« (Sieb des Eratosthenes) die Berechnung der Fakultät das Standardbeispiel für die Umsetzung rekursiver Algorithmen. Folgendes Programm berechnet Fakultäten im Basic 2.0 des C64 und ist also sehr nützlich für einige mathematische Funktionen:

Listing 2.30: EZ FAKULTAET

```
10 REM FAKULTAETEN
20 :
30 INPUTA:FORB=1TOA:C=C+LOG(B):NEXT:C=C/LOG(10):PRINT10↑(C-INT(C));"E";INT(C):RU
N


10 REM FAKULTAETEN
20

30 INPUTA
   FORB=1TOA
   C=C+LOG(B)
   NEXT
   C=C/LOG(10)
   PRINT10↑(C-INT(C));"E";INT(C)
   RUN
```

A ist dabei die Eingabe-, B die Zähl- und C- die Rechen-/Ausgabe-Variable. Das Programm berechnet sogar Fakultäten für größere Zahlen als 69; damit übertrifft es die meisten elektronischen Taschenrechner. Auf jeden Fall beweist es, daß der C64 für Basic-Zwecke offensichtlich doch nicht so ungeeignet ist, wie oft behauptet wird!
An dieser Stelle möchte ich jedoch kurz auf einen echten Leckerbissen für mathematische Zwecke hinweisen: Das »Scientific Basic«, welches von Heureka Teachware (Paul-Hösch-Str. 4, 8000 München 60, Tel.: 8 20 12 00) angeboten wird; normalerweise möchte ich keine Werbung für Produkte machen, doch ich empfehle denjenigen Lesern, die mathematische Pro-

blemstellungen komfortabel lösen möchten, sich über diese Neuentwicklung zumindest bei der genannten Adresse zu informieren; sie enthält die Fakultätsfunktion bereits als Funktion FACT. Mit Scientific Basic werden übrigens zahlreiche Anwendungen mitgeliefert, die schon für sich betrachtet hochinteressant sind (von Matrizeninversion bis Fractal-Grafik); der Funktionsplotter dieses Systems ist auch in der Lage, die Fakultätsfunktion grafisch darzustellen. Tatsache ist, daß Scientific Basic nicht nur alle Funktionen wissenschaftlicher Taschenrechner und herkömmlicher Grafikerweiterungen bietet, sondern auch viele Basic-Implementationen großer Rechner im mathematischen Bereich übertrifft. Leider ist es jedoch noch recht unbekannt, weshalb ich mich an dieser Stelle für dieses Programm einsetzen wollte.

Einzeiler #31: Der C64 als Mathe-Genie

Dieses Programm bedient sich des programmierten Direktmodus, um eingegebene mathematische Ausdrücke in Basic-Syntax auszuwerten. Sie geben eine Formel wie SIN(5)+47*COS(2) ein und erhalten sofort das Ergebnis geliefert.

Listing 2.31: EZ MATHE-GENIE

```
1 PRINT"(CLR)(LBLU)"A$"="A:INPUTA$:PRINT"(HOME)(BLU)A="A$":GΓ1:":POKE631,19:POKE
632,13:POKE198,2


1 PRINT"(CLR)(LBLU)"A$"="A
  INPUTA$
  PRINT"(HOME)(BLU)A="A$"
  GΓ1
  "
  POKE631,19
  POKE632,13
  POKE198,2
```

Damit läßt sich der C64 zum intelligenten Formel-Interpreter ausbauen. Vor allem für Programme im technischen Bereich bietet sich dieser Trick an; die zugrundeliegende Problemstellung ließe sich sonst nur in unzähligen Programmzeilen mit erheblichem Zeitaufwand bei Berechnung und Programmierung lösen.

Einzeiler #32: Zahlenrate-Spiel, Version 1

Das folgende Programm auf Seite 84 läßt Sie eine Zahl zwischen 0 und 999 erraten.
Dabei erteilt der C64 jeweils Auskunft, ob die von Ihnen geratene Zahl größer (1), kleiner (–1) oder gleich (0) der vom Computer erdachten Zahl ist. Sie haben bis zu 99 Versuche, die aktuelle Versuchsnummer erscheint hinter der Bewertung der letzten Eingabe. Nach richtigem Erraten wird eine neue Zahl ausgesucht.

Listing 2.32: EZ ZAHLENRATEN

```
7 PRINT"NEUE ZAHL":A=INT(RND(1)*10↑3):FORB=1TO99:INPUTC:D=SGN(A-C):PRINTD,B:OND+
1GOTO7:NEXT

7 PRINT"NEUE ZAHL"
  A=INT(RND(1)*10↑3)
  FORB=1TO99
  INPUTC
  D=SGN(A-C)
  PRINTD,B
  OND+1GOTO7
  NEXT
```

Interessant ist die Verwendung der SGN-Funktion; dabei wird der eingegebene Wert C vom richtigen Wert A abgezogen, und das Vorzeichen dieser Differenz ausgegeben. Dabei gibt es drei verschiedene Fälle:

1 $A > C \Rightarrow A-C > 0 \Rightarrow$ SGN-Wert = 1
2 $A = C \Rightarrow A-C = 0 \Rightarrow$ SGN-Wert = 0
3 $A < C \Rightarrow A-C < 0 \Rightarrow$ SGN-Wert = –1

Einzeiler #33: Zahlenrate-Spiel, Version 2

Die SGN-Ausgabe des letzten Einzeilers war nicht gerade das, was man sich unter »der menschlichen Ausdrucksweise angenähert« vorstellen mag. Damit räumt folgender Einzeiler von ähnlicher Struktur auf:

Listing 2.33: EZ ZAHLENRATEN 2

```
1 X=INT(RND(1)*100)+1:FORI=1TO1000:INPUTA:PRINTCHR$(61+SGN(A-X)):IFA< >XTHENNEXT

1 X=INT(RND(1)*100)+1
  FORI=1TO1000
  INPUTA
  PRINTCHR$(61+SGN(A-X))
  IFA< >XTHENNEXT
```

Er ermittelt eine Zahl zwischen 1 und 100, läßt Ihnen 1000 Versuche und – darin liegt nun der wichtige Unterschied – gibt nach jedem Versuch eines der folgenden Zeichen aus:

< eingegebene Zahl < Zahl des Computers
= eingegebene Zahl richtig
> eingegebene Zahl > Zahl des Computers

Das Funktionieren dieser recht plausiblen Ausgabeform liegt in der Formel

```
CHR$(61+SGN(A-X))
```

begründet: Die ASCII-Codes für »>«, »=« und »<« liegen im Bereich 60–62, welcher durch diese Formel erfaßt wird. Beispiel:

```
A=X ⇛ SGN(A-X)=0 ⇛ CHR$(61) ⇛ "="
```

Einzeiler #34: Primzahlen

Vorhin wurde bereits das »Sieb des Eratosthenes«, das gängige Verfahren zur Primzahlenberechnung, erwähnt. Hier ist ein Programm, das mit »Affentempo« die Primzahlen bis 1000 ausgibt:

Listing 2.34: EZ PRIMZAHLEN

```
10 DIMA(2000):FORI=2TO1000:IFA(I)=0THENPRINTI:FORJ=I+ITO1000STEPI:A(J)=1:NEXTJ,I
:01100 NEXT


10 DIMA(2000)
   FORI=2TO1000
   IFA(I)=0THENPRINTI
   FORJ=I+ITO1000STEPI
   A(J)=1
   NEXTJ,I
   01100 NEXT
```

Vor dem Start ist jedoch die Anweisung POKE2109,0 einzugeben. Dadurch wird der Einzeiler nämlich effektiv zu einem »Halb-Zweizeiler«. Die hohe Geschwindigkeit allerdings wurde vor allem durch STEP I erreicht, weil auf diese Weise ein Großteil der irrelevanten Zahlen von vornherein übersprungen wird, ohne daß eine zeitraubende Überprüfung anfällt.

Einzeiler #35: Wochentag

Ein reizvolles Thema ist auch die Ermittlung des Wochentags zu einem bekannten Datum. Eine Lösung für alle üblicherweise vorkommenden Daten bietet dieser Einzeiler:

Listing 2.35: EZ WOCHENTAG

```
1 INPUTT,M,J:T%=T+J+(J+(M<3))/4+3*M+2*(M>2)-INT((M-1+(M>8))/2)+2:PRINTT%-7*INT(T
%/7)
2 :
3 :
4 REM DATUM WANDELN
```

```
1 INPUTT,M,J
  T%=T+J+(J+(M<3))/4+3*M+2*(M>2)-INT((M-1+(M>8))/2)+2
  PRINTT%-7*INT(T%/7)

2

3

4 REM DATUM WANDELN
```

Er wünscht die Eingabe von Tag (T), Monat (M) und Jahr (J), letzteres mit Angabe des Jahrhunderts (z.B. auch 1987). Daraufhin erscheint eine Kennziffer für den Wochentag:

0	SUN	Sonntag
1	MON	Montag
2	TUE	Dienstag
3	WED	Mittwoch
4	THU	Donnerstag
5	FRI	Freitag
6	SAT	Sonnabend (Samstag)

Die verwendete Gleichung ist eine Vereinfachung der folgenden:

```
Z= T +365*J + INT ((J+(M<3))/4) + 31*M -31 + 2*(M>2) - INT
((M-1 + (M>8))/2))
```

Die erste INT-Funktion wird durch die Integervariable T% ersetzt. Weitere Vereinfachungen ergeben sich aus dem Gebrauch der Modulo-Funktion (Rest einer ganzzahligen Division), die in der PRINT-Anweisung benutzt wird. Es kommt durch Anwendung bestimmter Rechenregeln der MOD-Funktion zu folgenden Vereinfachungen:

365*J ergibt J, aus 31*M–31 wird 3*M.

Mit der jetzt erhaltenen Formel würde man schon auskommen. Um aber eine gebräuchliche Zahl-/Wochentags-Zuordnung zu erhalten, addieren wir noch 2 zu T%. Zur Erklärung der Gleichung: Es wird die Gesamtzahl aller Tage vom 0.0.0000 bis einschließlich des eingegebenen Datums berechnet. Nach Division durch 7 ergibt sich der gesuchte Wert aus dem ganzzahligen Wert. Der Rest wird in der PRINT-Anweisung durchgeführt (Modulo-Berechnung). Die Gleichung selbst ist folgendermaßen aufgebaut:

T	ist die Zahl der Tage im laufenden Monat
365*J	sind die Tage aller vorherigen Jahre
INT((3+(M<3))/4)	sind alle vorherigen Schalttage
31*M−31	sind die Tage aller vorherigen Monate, allerdings mit dem Fehler, daß einige Monate weniger als 31 Tage haben
2*(M>2)	es werden 2 abgezogen, wenn M>2 ist, d.h., der Februar wird zu einem Monat mit 30 Tagen gemacht
−INT((M−1+(M>8))/2)	zieht für jeden Monat mit 30 Tagen einen ab

Alle Teile werden letztlich addiert und nach obigem Schema umgeformt.

Einzeiler #36: Berechnung der Kreiszahl PI

Die Berechnung eines Näherungswertes für die Kreiszahl PI, eine Zahl mit unendlich vielen, nicht-periodischen Nachkommastellen, ist im Grunde kein technisches Erfordernis, da das PI-Symbol in Basic den Wert 3.14159265 hat. Der Algorithmus zur eigenen Ermittlung dieses Wertes ist jedoch eine recht interessante, wissenschaftliche Spielerei und wird von folgendem Einzeiler mit Bravour bewältigt:

Listing 2.36: EZ PI-BERECHNUNG

```
1 FORI=ØTO12:W=-(U=Ø):O=2*SQR(3)*W+O:U=3*W+U:O=2*O*U/(O+U):U=SQR(O*U):PRINT6*2↑I
,U:O:NEXT
2 :
3 REM ** BERECHNUNG DER KREISZAHL PI **

1 FORI=ØTO12
  W=-(U=Ø)
  O=2*SQR(3)*W+O
  U=3*W+U
  O=2*O*U/(O+U)
  U=SQR(O*U)
  PRINT6*2↑I,U;O
  NEXT
2

3 REM ** BERECHNUNG DER KREISZAHL PI **
```

Man kann mitverfolgen, wie der Wert immer genauer wird und auf das phänomenal exakte Ergebnis konvergiert. Vielleicht bildet dieses Programm ja eine gewisse Grundlage für die Diplomarbeit eines Lesers...

Einzeiler #37: Zugriffszeit der Floppy verkürzen

Der vorliegende Einzeiler dient dazu, die Zugriffszeit der Floppy 1541/70/71 drastisch zu verkürzen.

Listing 2.37: EZ FLOPPY-SPEEDER

```
10 OPEN1,8,15,"M-W"+CHR$(7)+CHR$(28)+CHR$(1)+CHR$(15)
20 :
30 REM ZUGRIFFSZEIT DER FLOPPY KUERZER
```

Der Schrittmotor, welcher den Schreib-/Lesekopf bewegt, kann erfahrungsgemäß wesentlich schneller arbeiten, ohne daß eine sichere Funktion der Floppy gefährdet wird. Da der Schrittmotor im Interrupt der Floppy bedient wird, genügt es zur Beeinflussung der Drehzahl des Motors, die Größe des Interrupt-Intervalles zu verändern. Standardmäßig wird etwa alle 15 Millisekunden ein Interrupt ausgelöst, der den Stepper um eine Viertelspur weiterbewegt. Durch das vorliegende Programm wird diese Zeit auf etwa 4 Millisekunden verkürzt. Alle Bewegungen des Kopfes werden dadurch fast viermal schneller. Dies wiederum hat neben der Zeitersparnis noch zwei weitere Vorteile von eminenter Wichtigkeit: Das Laufgeräusch des Kopfes wird angenehm leise und kurz, und im Falle einer Kopfjustage (MG-salvenartiges Geräusch) fährt der Kopf mit erheblich verminderter Kraft gegen den Anschlag, so daß die Gefahr einer Dejustage deutlich gemindert ist.
Um jedoch Mißverständnissen vorzubeugen: Das eigentliche Laden oder Abspeichern eines Programms wird nicht schneller. Aber schon mit bloßem Auge läßt sich erkennen, daß nach Ausführung von Einzeiler #37 die Zeit zwischen der Ausgabe von »SEARCHING FOR xxx« und »LOADING« um ein Vielfaches schneller ist. Vor allem das lange Directory der Masterdiskette ist ebenfalls ein entscheidender Faktor für die effektive Ladezeit einer Datei, und kann durch Erhöhung der Zugriffsfrequenz deutlich beschleunigt werden. Dies funktioniert jedoch mit manchen Floppy-Speedern nicht.

Einzeiler #38: Do-it-yourself-Justage der Floppy

Um Disketten von verstellten Laufwerken zu lesen, genügt folgender Einzeiler:

Listing 2.38: EZ JUSTAGE

```
10 OPEN1,8,15,"M-W"+CHR$(105)+CHR$(0)+CHR$(X):CLOSE1
20 REM [X] IST DIE ANZAHL DER ZUGRIFFE


10 OPEN1,8,15,"M-W"+CHR$(105)+CHR$(0)+CHR$(X)
   CLOSE1
20 REM [X] IST DIE ANZAHL DER ZUGRIFFE
```

Dieses Programm schreibt in die Speicherzelle $6A des Floppy-Speichers den Wert X, der vorher mit »X=« zu definieren ist: Wird diese Konstante (Standardwert 5) größer gewählt, so kann man auch Disketten lesen, die dem Laufwerk zuvor Probleme bereiteten. Die Zahl X ist nämlich die Anzahl der Leseversuche, die vor Feststellung eines Diskettenfehlers unternommen werden. Die Zugriffszeit erhöht sich natürlich entsprechend der Konstanten und der Anzahl der auftretenden Leseschwierigkeiten. Man verfügt aber dadurch über die Möglichkeit, auch Disketten von schlecht justierten Laufwerken zu lesen, um die darauf enthaltenen Dateien zu retten.

Einzeiler #39: Absoluter Datenschutz für Disketten

Datenschutz im doppelten Sinne ermöglicht dieser Einzeiler, der niemals auf Disketten angewendet werden sollte, von denen Sie keine Sicherheitskopie mehr haben:

Listing 2.39: EZ PROTECTOR

```
1 OPEN1,8,3,"#":OPEN2,8,15,"B-P3,144":PRINT#1,"(DEL)(DEL)(DEL)"CHR$(0)CHR$(0)CHR
$(0):PRINT#2,"U2:3,0,18":PRINT#2,"I
2 REM
3 REM VERHINDERT LISTEN DES DIRECTORY
4 REM VON BASIC AUS  UND BRINGT EINEN
5 REM SCHREIBSCHUTZ AUF DIE DISKETTE.



1 OPEN1,8,3,"#"
  OPEN2,8,15,"B-P3,144"
  PRINT#1,"(DEL)(DEL)(DEL)"CHR$(0)CHR$(0)CHR$(0)
  PRINT#2,"U2
  3,0,18"
  PRINT#2,"I
2 REM
3 REM VERHINDERT LISTEN DES DIRECTORY
4 REM VON BASIC AUS  UND BRINGT EINEN
5 REM SCHREIBSCHUTZ AUF DIE DISKETTE.
```

Datenschutz 1: Das Directory wird nicht mehr vollständig gelistet.

Datenschutz 2: Die Diskette kann weder beschrieben noch gelöscht oder irgendwie verändert werden. Auch Diskettenbefehle wie VALIDATE werden ignoriert. Sogar ein Formatieren ohne ID ist verhindert. Das einzige, was den Inhalt der Diskette noch manipulieren kann, ist das Formatieren (Diskettenbefehl N) mit einer neuen ID.

Beispiel:

```
OPEN1,8,15,"N:LEER,LR"
```

Das Laden der bereits auf der Diskette befindlichen Programme funktioniert hingegen völlig normal. Allerdings sollte man sich unbedingt die Namen der gespeicherten Files merken, denn

das intuitive Suchen mit Jokerzeichen ist bestenfalls ein netter Zeitvertreib, aber nicht immer erfolgreich (man denke nur an das Problem, daß sich zwei Programme, die mit A anfangen, auf derselben Diskette befinden, weshalb man auch den zweiten Buchstaben zur Erkennung des zweiten Programms wissen muß...). Daß das Directory nicht mehr gelistet werden kann, liegt daran, daß es als Basic-Programm geladen wird und in dem veränderten Directory die Basic-Endmarkierung (dreimal $00) am Anfang steht. Das Directory-Programmfile »$« endet also bereits nach zehn Byte. Die ersten fünf Byte stellen Programmanfang und Zeilennummer dar; darauf folgen Leerzeichen und Anführungszeichen als konstanter Anfang eines Inhaltsverzeichnisses. Um selbst diese drei übrigen Zeichen beim Auflisten verschwinden zu lassen, folgen nun drei CHR$(20), die jeweils ein DELETE darstellen. Diese wurden im Insert Mode eingegeben, werden aber beim LISTEN ausgeführt und bringen am Bildschirm – im Gegensatz zum Drucker – ein etwas chaotisches Listing.
Nach diesen DELETEs endet das Listen, da unmittelbar die drei Nullen folgen, welche das Programmende markieren. So erscheint nach LIST insgesamt gesehen nur die READY-Meldung – Sie wissen jetzt, wie es im Detail zu diesem Phänomen kommt.
Das Schützen der Diskette vor Schreibzugriffen geschieht als »Abfallprodukt«, da das dritte Byte auf Block 18/0 mit einem anderen Wert als $41 (ASCII-Code für »A«) belegt wird. Der eigentliche Zweck des Einzeilers besteht jedoch im Directory-Listschutz.

Einzeiler #40: Kopierschutz

Diese Zeile produziert auf dem gewünschten Track der Diskette (Variable T entsprechend belegen!) einen READ ERROR 21:

Listing 2.40: EZ TRACK-KILLER

```
1 OPEN1,8,15:OPEN2,8,2,"#":PRINT#1,"U1 2 0";T;0:PRINT#1,"M-E"CHR$(163)CHR$(253)
2 :
3 REM *** TRACK-KILLER ***
4 REM ERZEUGT AUF SPUR [T] EINEN
5 REM READ ERROR 21 (NO SYNC)
6 :


1 OPEN1,8,15
  OPEN2,8,2,"#"
  PRINT#1,"U1 2 0";T;0
  PRINT#1,"M-E"CHR$(163)CHR$(253)
2

3 REM *** TRACK-KILLER ***
4 REM ERZEUGT AUF SPUR [T] EINEN
5 REM READ ERROR 21 (NO SYNC)
6
```

Dessen Abfrage könnte einen einfachen, aber wirkungsvollen Kopierschutz bedeuten. Sollten Sie allerdings nicht dieses Ziel im Auge und eine ansonsten unbenötigte Diskette zur Hand haben, lassen Sie lieber von der Verwendung des Programms ab; es kann die Daten kompletter Disketten unwiederbringlich zerstören!

Einzeiler #41: Abfrage eines Gerätes

Ein recht großes Problem stellt in vielen Programmen die Erkennung dar, ob ein Gerät eingeschaltet und verfügbar ist. Wirklich bedienungsfreundliche Programme sollten über eine solche Kontrolle verfügen:

Listing 2.41: EZ CHECKDEVICE

```
1 OPEN1,G,S:CLOSE1:IF ST<0 THEN PRINT"FEHLER":STOP
2 :
3 REM [G] = GERAETEADRESSE
4 REM [S] = SEKUNDAERADRESSE
```

Dadurch werden unnötige DEVICE NOT PRESENT ERRORs vermieden, die den Anwender oftmals die Mühe stundenlanger Arbeit kosten können. Die zu prüfende Geräteadresse muß in G, die Sekundäradresse in S übergeben werden. Bei der Floppy bietet sich G=8 und S=15, beim Drucker G=4 und S=0 an. Das verwendete Testverfahren hat keine unerwünschten Nebenwirkungen auf die Peripheriegeräte.

Einzeiler #42: Tastaturabfrage mit ON-GOTO-Sprung

Bei Abfrage mehrerer Tasten in einem Menü setzt man normalerweise eine oft unüberblickbare Kolonne von IF-THEN-GOTOs ein. Es geht jedoch auch mit einer genialen Technik viel besser:

Listing 2.42: EZ TASTENABFRAGE

```
1 REM TASTATURABFRAGE MIT SPRUNG
2 :
500 GETA$:ON1-(A$="A")-2*(A$="B")-3*(A$="X")GOTO500,2000,3000:END
600 :
610 :
2000 PRINT"TASTE <A> GEDRUECKT.":GOTO500
3000 PRINT"TASTE <B> GEDRUECKT.":GOTO500

1 REM TASTATURABFRAGE MIT SPRUNG
2

500 GETA$
    ON1-(A$="A")-2*(A$="B")-3*(A$="X")GOTO500,2000,3000
    END
600

610
```

```
2000 PRINT"TASTE <A> GEDRUECKT."
     GOTO500
3000 PRINT"TASTE <B> GEDRUECKT."
     GOTO500
```

Dieses Programm wartet auf die Tasten <A> und <B>, welche eine eigene Sonderbehandlung (Ausgabe eines bestimmten Textes) haben; bei <X> wird das Programm beendet. Der eigentliche Sprungverteiler nimmt jetzt weniger Raum ein und ist aufgrund der gesammelten Sprungziel-Liste hinter GOTO letztlich aufschlußreicher, als wenn mehrere IF-THEN-GOTOs zu analysieren wären.

Das Prinzip, auf dem der ON-Ausdruck basiert, wird in 2.6 ausführlich erläutert. Zum Verständnis sei nur gesagt, daß ein Ausdruck wie (A$= "X") den Wert 0 liefert, wenn der Ausdruck nicht stimmt, andernfalls –1. Dadurch ergeben alle Subtraktionen falscher Ergebnisse (–A$= "falscher Tastendruck") den Wert 0, verändern also das Gesamtergebnis nicht; an der richtigen Stelle wird jedoch –1*x subtrahiert (–1 für den richtigen Ausdruck, X für die Nummer des Menüpunktes), also letztlich der Wert »x« addiert.

Für das Beispiel A$= "B" wollen wir nun den ON-GOTO-Ausdruck berechnen:

1–(A$= "A")	–2*(A$= "B")	–3*(A$= "X")
1–0	–2*(–1)	–3*(0)
1	–(–2)	–0
1	+ 2	=3

Wie man sieht, ist ab dem dritten Sprungziel (3000) die entsprechende Routine vorhanden. Die erste Zeilennummer hinter ON-GOTO ist hier jeweils die GET-Zeile; dadurch wird solange gewartet, bis eine der gewünschten Tasten gedrückt wird. Auch für das gezielte Warten auf einen oder mehrere Tastendrücke bietet sich die beschriebene Technik an.

Einzeiler #43: Grafikbildschirm mit DIM-Befehl löschen

Das Löschen einer Hires-Bitmap über eine FOR-NEXT-Schleife ist für diejenigen Grafikprogrammierer, die sich keiner Grafikerweiterung bedienen (obwohl es diese wie Sand am Meer gibt), eine nervenzerreißende Geduldsprobe. Man sieht förmlich, wie jede 8-Punkte-Einheit gelöscht wird. Dennoch gibt es bestimmte Gründe, auch einmal über PEEK/POKE die Hires-Grafik zu programmieren (siehe »Alles über den C64«). Hier nun ein cleverer Lösungsweg für ein schnelles HIRES CLR:

Listing 2.43: EZ GRAFIK-CLR

```
10 A=0:B=0:A=PEEK(49):B=PEEK(50):DIMF((16191-A-B*256)/5):POKE49,A:POKE50,B
```

```
10 A=0
   B=0
   A=PEEK(49)
   B=PEEK(50)
   DIMF((16191-A-B*256)/5)
   POKE49,A
   POKE50,B
```

Hier nun die Programmbeschreibung zu diesem Trick, der zugegebenermaßen nur für sehr versierte Leser seinen hohen Wert hat:

```
1. a=0:b=0
```

Im weiteren Verlauf wird das Variablenfeld beeinflußt. Die Variablen a und b müssen deshalb vor der DIM-Anweisung angelegt sein. Würden a und b hier nicht belegt, so würde in Schritt #2 der Zeiger 49/50 nicht auf diese Variablen vorbereitet sein – weil der Basic-Interpreter sie ja erst neu definiert – und die Folge wäre eine verheerende »Speicherverirrung«.

```
2. a=peek(49):b=peek(50)
```

Die Werte für das Variablenende im Speicher (siehe »C64 für Insider«) werden gesichert.

```
3. dim f((16191-a-b*256)/5)
```

Durch das Dimensionieren einer Variablen wird ein entsprechend großer Platz hinter dem bisherigen Variablenende angelegt und mit Nullbytes gelöscht.
Zur Formel (16191-a-b*256)/5:
Die zu löschende Bitmap muß im Bereich 8192–16191 liegen. Die Variable f wird hinter dem bisherigen Variablenende angelegt. Die Adresse des Variablenendes ist a+b*256 und muß daher subtrahiert werden. Pro indizierte Variable werden 5 Byte gelöscht, daher die Division durch 5:

```
(16191-a-b*256)/5
```

Die mit 0 indizierte Variable ist nicht berücksichtigt, ebenso die ersten 7 Byte für Variablenname und Dimension; diese Bytes erhalten nämlich Werte < > 0. Zum Verständnis des DIM-Befehls werden Sie mit bisherigen ROM-Listings und ähnlichen Büchern mit Sicherheit nicht auskommen; diese drücken sich in der Regel um die entscheidenden Programmstrukturen und Details in der wirklich nicht einfach angelegten DIM-Routine. Hier benötigen Sie eine komplette Dokumentation aus detailliertem ROM-Listing und Systemdokumentation mit Flußdia-

gramm (siehe »C64 für Insider«); darin befindet sich auch eine Beschreibung des kompletten Aufbaus von Variablen und Variablenfeldern, Zahlen- und Stringformaten mit zahlreichen veranschaulichenden Abbildungen.

4. poke 49,a:poke 50,b

Die alten Werte werden wieder hergestellt. Die Variable f ist jetzt nicht mehr dimensioniert, da ihr Speicherbereich quasi »von der Außenwelt abgeschnitten wird«. Für den C64 läuft es jetzt so weiter, als wäre f() nie dimensioniert worden. Das zuvor erfolgte Bereichslöschen jedoch bleibt erhalten.

5. Hinweise

- Das Variablenende muß < 8186 sein, d.h., das Programm darf einschließlich Variablenfeld nicht die Grenze von 6 Kbyte überschreiten. Damit steht jedoch meist ein ausreichend großer Platz zur Verfügung. Wird die 6-Kbyte-Grenze nicht eingehalten, so äußert sich dies lediglich in einem unvollständigen oder im Extremfall – ausbleibenden – Bildschirmlöschen; das Programm nimmt jedoch keinesfalls Schaden, weil der Zeiger 49/50 niemals auf einen Speicherbereich innerhalb des Programms weist.
- Soll das Grafikprogramm zum Abspeichern von Bildern benutzt werden, darf man nicht vergessen, den Zeiger für die Speichergrenze (55/56) auf eine Adresse kurz hinter dem Bildschirmspeicher zu positionieren.
- Das beschriebene Verfahren setzt voraus, daß bereits eine Hires-Bitmap ab $2000 (#8192) eingeschaltet wurde. Wie dies im einzelnen geht, erfahren Sie in »C64 – Wunderland der Grafik« oder in »Alles über den C64«.
- Dennoch sollte dieser Einzeiler mehr wegen der tollen Idee, auf welcher er basiert, abgedruckt werden; zudem fördert er das Verständnis für die dynamische Variablenverwaltung. Vorzuziehen ist in jedem Fall die Verwendung eigener Maschinenroutinen oder – für Basic-Zwecke – eine leistungsstarke Grafik-Befehlserweiterung.

Einzeiler #44: Grafikbildschirm invertieren

Dieser Einzeiler invertiert eine Hires-Grafik-Bitmap an der Adresse, welche in der Variablen A steht:

Listing 2.44: EZ HIRES-INVERS

```
1 A$="(HOME)¬(RVS)"■(OFF) ":PRINTA$:FORI=0TO8191:POKE780,PEEK(A+I):SYS1024:POKEA+
I,PEEK(780):NEXT
10 :
20 REM  HIRES-INVERS

1 A$="(HOME)¬(RVS)"■(OFF) "
  PRINTA$
  FORI=0TO8191
```

```
  POKE780,PEEK(A+I)
  SYS1024
  POKEA+I,PEEK(780)
  NEXT
10

20 REM  HIRES-INVERS
```

Um das besser verstehen zu können, sollten Sie eine Hires-Grafik laden. Nehmen wir als Beispiel ein Bild aus dem Zeichenprogramm Hi-Eddi (plus) und laden es beispielsweise in Bildschirm 1. Verlassen Sie jetzt Hi-Eddi (plus), und geben Sie den Einzeiler ein. Weisen Sie der Variablen A den Wert des Grafik-Anfangs (meist 8192) zu, und springen Sie mit GOTO 1 in den Einzeiler. Die Grafik wird jetzt also blitzschnell invertiert; Sie können Hi-Eddi (plus) erneut laden und den Bereich 8192–16384 als invertierte Grafik abspeichern.
Erklärung dieser Zeile, die wiederum vor allem eine wertvolle Anregung darstellt (schließlich hat Hi-Eddi eine eigene Invertier-Routine!):
Der String A$ enthält ein Maschinenprogramm, welches durch den PRINT-Befehl in den Bildschirmspeicher kommt. Die aktuelle Speicherzelle der Bitmap, welche invertiert werden soll (was durch EOR- # $FF-Verknüpfung geschieht), wird durch POKE 780,PEEK(A+I) im Akku abgelegt. Nach SYS1024 steht dann der Wert wiederum in 780 und wird mit dem anschließenden POKE in die entsprechende Speicherstelle als invertiertes Bitmuster gebracht.
Noch eine Bemerkung: Die Grafik darf natürlich nicht unter einem ROM liegen, da dann die Aufgabenstellung nur in reiner Maschinensprache zu lösen wäre (das ROM müßte abgeschaltet werden). Das Maschinenprogramm im Bildschirmspeicher besteht nur aus zwei Befehlen:

```
,0400     EOR #$FF  ; = %11111111 = Invertierung aller Bits
,0402     RTS
```

Einzeiler #45: Rasterinterrupt in Basic?

Eine Domäne der Maschinenprogrammierung ist die sogenannte Unterbrechungsprogrammierung (Interrupt). Unvergessen sind die zahlreichen Effekte, die Assembler-Künstler damit auf den Bildschirm gezaubert haben (Sprites außerhalb des eigentlichen Bildschirms, mehrfarbiger Bildschirmrahmen, Text und Grafik in gemischter Darstellung, ...). Sollte es auch in Basic möglich sein?

Listing 2.45: EZ RASTER-IRQ

```
10 POKE53280,1      ::::::::::::::::POKE53280,0      :::::::::::::::GOTO10
20 REM *** ZWEIFARBIGER BILDSCHIRM ***
```

Natürlich nicht; hier entscheidet nur das richtige Timing beim Umschalten zwischen zwei Bildschirmfarben über den Raster-IRQ-ähnlichen Effekt. Die Leerzeichen und Doppelpunkte dürfen nicht verändert werden, weil sonst die ganze Wirkung verlorengeht. Die ausgewogene Verzögerung durch Leerzeichen und Doppelpunkte innerhalb der Endlosschleife ist nämlich für das Funktionieren dieses Einzeilers verantwortlich. So einfach es auch sein mag: Es verblüfft einen doch!

Einzeiler #46: Neuer Zeichensatz

Abwechslung ist alles. Das folgende Programm enthält nur eine Variable, gerade sechs POKE-Befehle und jeweils nur einen PEEK und eine FOR-NEXT-Schleife, doch es ist in der Lage, den Zeichensatz des C64 völlig neu zu gestalten:

Listing 2.46: EZ NEWFONT

```
1 R=56334:POKE53272,24:POKER,0:POKE1,51:FORI=6↑5TOR/5:POKEI,PEEK(I+45056)AND60:N
EXT:POKE1,7:POKER,1
2 :
3 REM *** NEWFONT: NEUER ZEICHENSATZ ***



1 R=56334
  POKE53272,24
  POKER,0
  POKE1,51
  FORI=6↑5TOR/5
  POKEI,PEEK(I+45056)AND60
  NEXT
  POKE1,7
  POKER,1
2

3 REM *** NEWFONT
   NEUER ZEICHENSATZ ***
```

Eine Warnung vorweg: Die Programmausführung dauert recht lange; unterbrechen Sie es nicht, sonst stürzt es möglicherweise ab. Doch das Warten lohnt sich!

Nun zur Funktionsweise dieses Wahnsinnsprogramms:
In der Variablen R wird die Adresse des Interrupt-Kontrollregisters (#56334 = $dc0e) definiert. Nun wird die Position des Zeichensatz-Speichers geändert, indem wir Bit 3 der Adresse 53272 ($d018 = VIC-Register #24) setzen und Bit 1 löschen; so wird der Bereich ab #8192 ($2000) ausgewählt. Um den alten Zeichensatz noch lesen zu können, gehört der Interrupt ausgeschaltet (Bit 0 im Kontrollregister R wird gelöscht) und gleichermaßen der Video-Chip (Bit 2 in Adresse 1 gelöscht). Die sich anschließende Schleife würde in einem unter Normalbedingungen zustandegekommenen Programm folgendermaßen aussehen:

```
FOR I=0 TO 4095
POKE 8192+I,PEEK(I+53248) AND 60
NEXT
```

Es wird also der Zeichensatz von Adresse #53248 ($d000) nach #8192 ($2000) übertragen, wobei jedes Byte durch die AND-Funktion verändert wird; diese bewirkt, daß die Bits 0, 1, 6 und 7 in jedem Byte gelöscht werden, also effektiv jedem Zeichen der rechte und linke Rand abgeschnitten wird. In diesem Programm überdeckt die Schleife einen etwas größeren Bereich als eigentlich nötig, nämlich von 6^5 = 7776 bis R/5=11266; dies tut dem Resultat jedoch keinen Abbruch und trug sehr zur Verkürzung der Eingabe bei – schließlich sollte ja das Ganze in eine einzige Zeile passen!
Zuletzt werden Video-Chip und Interrupt wieder eingeschaltet (Bit 2 in Adresse 1, Bit 1 in Adresse 56334 gesetzt).
Bemerkenswert ist vielleicht, daß Klaus Vorwalter, der Autor dieses Programms, nach Realisierung seiner Idee ein einziges Zeichen zuviel im Programm hatte, als daß es in eine einzige Zeile gepaßt hätte. Nach drei Stunden angestrengten Tüftelns kam ihm jedoch die erlösende Idee. Ursprüngliche hatte der den Video-Chip durch POKE1,55 wieder eingeschaltet; es genügte aber letztlich doch POKE1,7, da die übrigens Bits vom Prozessor automatisch gesetzt werden (siehe »C64 für Insider«).

Einzeiler #47: Klavierton

Viele Programmierer, vor allem die Entwickler von Anwendungsprogrammen, interessieren sich herzlich wenig für die Welt des programmierten Sounds – wahrscheinlich auch deshalb, weil dieses Thema recht anspruchsvoll ist und viel Hingabe verlangt. Dennoch bietet sich eine gewisse akustische Untermalung selbst für »ernste« Programme an. So gibt beispielsweise die Textverarbeitung MasterText 128, mit der ich gerade schreibe, nicht nur bei jedem Tastendruck, sondern auch bei jeder außergewöhnlichen Situation wie dem Rücksprung ins Hauptmenü, einen dafür vorgesehenen Ton aus. Dadurch wird die gesamte Wahrnehmungsfähigkeit des Menschen zum besseren Umgang mit dem Computer eingesetzt.
Kurz und gut, es fehlt den meisten Programmierern nur an fertigen Routinen. Hier sei zunächst ein Klavierton als Anregung vorgestellt:

Listing 2.47: EZ KLAVIERTON

```
10 S=54272:POKES+24,15:POKES+1,110:POKES+5,9:POKES+6,9:POKES+4,17:POKES+4,16

10 S=54272
   POKES+24,15
   POKES+1,110
   POKES+5,9
   POKES+6,9
   POKES+4,17
   POKES+4,16
```

Die Variable S enthält die Basisadresse des Sound-Chips SID. Durch Experimentieren mit den Parametern der POKE-Befehle (also mit den Werten rechts vom Komma!) finden Sie bestimmt weitere Töne heraus – und sei es auch nur der Zufall, der Ihr musikalisches Talent weckt!

Einzeiler #48: Signalton

Folgender Ton eignet sich ebenfalls zur Einbindung in eigene Programme:

Listing 2.48: EZ SIGNALTON

```
10 S=54272:FORI=ØTO6:READX::POKES+I,X:NEXT:POKES+24,15:POKES+4,17:POKES+4,16:DAT
A,99,,8,,11,11

10 S=54272
   FORI=ØTO6
   READX

   POKES+I,X
   NEXT
   POKES+24,15
   POKES+4,17
   POKES+4,16
   DATA,99,,8,,11,11
```

Eine Aufspaltung in eine Zeile mit POKE-Befehlen und eine reine DATA-Zeile bietet sich an, da Sie sonst bei der Eingabe (und Veränderung!) der Zeile die Abkürzungen der Befehle benötigen. Auch für diesen Sound-Einzeiler gilt: Probieren geht über Studieren!

Einzeiler #49: Grafik-Power für den Plotter

Dieses Programm malt mit dem Plotter 1520 eine Spirale, die aus lauter Dreiecken entsteht:

Listing 2.49: EZ PLOTTER

```
1 OPEN1,6,1:PRINT#1,"M",240,Ø:FORI=1TO25Ø:PRINT#1,"S",24Ø+SIN(I*2)*I,COS(I*2)*I:
NEXT

1 OPEN1,6,1
  PRINT#1,"M",240,Ø
  FORI=1TO25Ø
  PRINT#1,"S",24Ø+SIN(I*2)*I,COS(I*2)*I
  NEXT
```

Zugrunde liegt eigentlich die Berechnung eines Kreises, da die X-Koordinaten mit SIN und die Y-Koordinaten mit COS ermittelt werden. Das Resultat ist jedoch eine Spirale, weil X und Y mit der Laufvariablen I multipliziert werden. Der Faktor »2« innerhalb eines SIN-/COS-Argumentes läßt sich gegen einen anderen Wert vertauschen; dadurch ändert sich auch das Aussehen der Grafik in gewissen Grenzen.

Einzeiler #50: Hardcopy

Dieser Einzeiler von Karl-Heinz Heß ist einer der allerbesten.

Listing 2.50: EZ HARDCOPY

```
1 REM *** HARDCOPY ***
2 :
6 REM *** AUFRUF: ***
7 PRINT"(HOME)";:OPEN1,4,7:OPEN2,3:GOSUB9:CLOSE2:CLOSE1:END
8 REM ** EIGENTLICHER EINZEILER: ***
9 FORI=1TO25:FORJ=1TO20:GET#2,A$:PRINT"(DEL)";:GET#2,B$:PRINT"(LEFT)(INST)"A$B$;
:PRINT#1,A$B$;:NEXT:RETURN
10 :
```

```
1 REM *** HARDCOPY ***
2

6 REM *** AUFRUF
   ***
7 PRINT"(HOME)";
  OPEN1,4,7
  OPEN2,3
  GOSUB9
  CLOSE2
  CLOSE1
  END
8 REM ** EIGENTLICHER EINZEILER
   ***
9 FORI=1TO25
  FORJ=1TO20
  GET#2,A$
  PRINT"(DEL)";
  GET#2,B$
  PRINT"(LEFT)(INST)"A$B$;
  PRINT#1,A$B$;
  NEXT
  RETURN
10
```

Er gibt den Text einer Bildschirmseite des C64 auf einem gängigen Drucker aus. Ganz ohne PEEKs, POKEs und SYS-Befehle. Trotzdem überrascht das Programm mit einer hohen Druckgeschwindigkeit.

Mittlerweile existiert eine Vielzahl von Hardcopy-Routinen für die verschiedenen Drucker und Bildschirmformate. Bisher gab es aber kein Programm, das universell einsetzbar ist, keine PEEKs und POKEs braucht und vor allem nicht in Maschinensprache geschrieben ist. Diese Eigenschaften weist der Einzeiler in Zeile 9 des Listings auf. Das »Programm« enthält nur Basic-Befehle. Die Programmidee basiert auf der oftmals vergessenen Tatsache, daß der Bildschirm des C64 wie eine Datei behandelt werden kann. Ähnlich wie der Drucker die Adresse 4 oder 5 hat und die Floppy mit Adresse 8 angesprochen wird, ist dem Bildschirm die Geräteadresse 3 zugeordnet, und zwar fest vom Betriebssystem (siehe »C64 für Insider«). Mit OPEN 2,3 wird die Bildschirmdatei eröffnet und mit INPUT #2 oder GET #2 der Inhalt der Datei gelesen. Beim INPUT #-Befehl ist, wie beim normalen INPUT auch, die Restriktion zu beachten, daß bei einem Komma oder Doppelpunkt abgebrochen wird. Deshalb wurde der INPUT- durch den GET-Befehl ersetzt.

Eine GET-Schleife liest ab der aktuellen Cursorposition sämtliche Zeichen als ASCII-Code. Die gelesenen Codes addiert man darauf in einer Variablen, z.B. durch A$=A$+B$, wobei immer wieder ein neues B$ zu A$ addiert wird. Eine einfache Sache sollte man meinen, aber die eigentlichen Schwierigkeiten beginnen erst an dieser Stelle. GET liest nämlich immer als letztes Zeichen einer Zeile CHR$(13), den Carriage-Return-Code – unabhängig davon, welches Zeichen sich dort befindet. Ferner ist der Editor des C64 zu beachten, welcher logische Zeilen von bis zu 80 Zeichen bildet, obwohl auf dem Bildschirm eine 40-Zeichen-Darstellung vorliegt. Nach dem Öffnen der Bildschirmdatei wird Zeichen für Zeichen gelesen, beginnend in der linken oberen Bildschirmecke. Dabei werden in einem Schleifendurchgang immer zwei Zeichen geholt und den Variablen A$ und B$ zugeordnet. Ist A$ gelesen, wird mittels DELETE = CHR$(20) der restliche Zeileninhalt um eine Stelle linksverschoben, um das letzte Zeichen einer Zeile eindeutig zu identifizieren.

Nach dem Einlesen von B$ wird durch die INSERT-Funktion CHR$(148) der ursprüngliche Zeilenabstand hergestellt. Anschließend wird der ASCII-Code in den beiden Variablen an den Drucker gesandt. Durch ständige, schnell ablaufende Zeilenverschiebung ist deutlich sichtbar, wo der Computer jeweils »arbeitet« – ein brauchbarer Nebeneffekt, finde ich.

Die leichten Unzulänglichkeiten:

Ein Einzeiler kann natürlich kein perfektes Programm sein. Deshalb sollten auch die kleinen Nachteile herausgestellt werden.

- Die Routine kann keine reversen Zeichen auf dem Drucker darstellen. Reverse Zeichen werden normal gedruckt.
- Innerhalb des Textes, von dem eine Hardcopy gedruckt werden soll, dürfen keinerlei Anführungszeichen vorkommen. Da CHR$(34) bei Bildschirmausgaben eines Programms recht selten auftritt, dürfte diese Einschränkung nur geringfügige Bedeutung haben. Anders verhält es sich bei Programmlistings, welche aber über OPEN1,4:CMD1:LIST gedruckt werden.

- Der vom Programm verwendete DELETE wurde im Insert Mode eingegeben und beim Listen am Bildschirm ausgeführt; dadurch verunstaltet er beim Bildschirm-LIST die Ausgabe, was zur verstümmelten Version PRINT"; führt.

Der Einzeiler ist dazu gedacht, als Unterprogramm Daten auszudrucken; einfach vom Bildschirm weg.

Aufruf der Routine:

Zeile 7 ist ein Beispiel für den Routinenaufruf. Zunächst wird der Cursor in die HOME-Position gebracht und mit OPEN1,4,7 die Drucker-, mit OPEN2,3 die Bildschirmdatei eröffnet. GOSUB 9 startet schließlich das Hardcopy-Unterprogramm. Nach dem Druck werden die beiden Dateien wieder mit CLOSE geschlossen.

Einzeiler #51: Der aufgeregte Marsmensch

Dieser Unterhaltungs-Einzeiler wurde eigentlich für den VC 20 entwickelt, läuft aber auch auf dem C64 und soll Ihnen wegen seines lustigen Effekts nicht vorenthalten werden:

Listing 2.51: EZ MARSMENSCH

```
1 PRINT"(HOME) ▝▁▁▖ (HOME)(DOWN)  (RVS)(DOWN)◤ ◥":PRINT"o(RVS) ▪ ▪ (OFF)o":PRINT"  (R
VS) ▬ ":PRINT" ◥(RVS)¯(OFF)◤":PRINT" ╱ ╲":POKE37877,0:PRINT"(HOME) ▂▂╱▪ (HOME)(DO
WN)(DOWN)((RGHT)(DOWN)(DOWN)(RVS)  ":RUN
10 :
20 REM  MARSMENSCH
```

```
1 PRINT"(HOME) ▝▁▁▖ (HOME)(DOWN)  (RVS)(DOWN)◤ ◥"
  PRINT"o(RVS) ▪ ▪ (OFF)o"
  PRINT"  (RVS) ▬ "
  PRINT" ◥(RVS)¯(OFF)◤"
  PRINT" ╱ ╲"
  POKE37877,0
  PRINT"(HOME) ▂▂╱▪ (HOME)(DOWN)(DOWN)((RGHT)(DOWN)(DOWN)(RVS)  "
  RUN
10

20 REM  MARSMENSCH
```

Im wesentlichen werden die Grafikzeichen der Tastatur sowie die Steuerzeichen im Quote Mode genutzt. Der POKE-Befehl wirkt nur auf dem VC 20 und erzielt dort eine Verlangsamung der Bildschirmausgabe.

Einzeiler #52: Das Labyrinth

Eine ähnliche Richtung wie Einzeiler #51 schlägt auch die Zeile auf der folgenden Seite ein:

Listing 2.52: EZ IRRGARTEN

```
1 X=RND(1)*2:PRINTCHR$(164+X);:Q=Q+1:IFQ<880THEN1
2 GOTO2:REM *** BILD ANHALTEN
3 :
4 REM *** IRRGARTEN MIT CHR$-CODES ***
```

```
1 X=RND(1)*2
  PRINTCHR$(164+X);
  Q=Q+1
  IFQ<880THEN1
2 GOTO2
  REM *** BILD ANHALTEN
3

4 REM *** IRRGARTEN MIT CHR$-CODES ***
```

Finden Sie einen Ausweg aus dem zufällig erzeugten Irrgarten? Auf jeden Fall sollten Sie anstelle von 164 auch andere Werte hinter CHR$() ausprobieren.

Einzeiler #53: Englisch-Kurs

Folgendes Listing wirkt alles andere als sinnvoll:

Listing 2.53: EZ ENGLISCH

```
0 REM">> ◥ ┌S ▌O ▬ ▬ITI╎ ╘ ╎ ♦ ▁ ╱ ◤ <<
```

Fahren Sie jetzt mit dem Cursor auf das am weitesten links stehende »>«, also genau rechts vom Anführungszeichen. Löschen Sie dann das Anführungszeichen mit <DEL>, drücken Sie <RETURN> und – abrakadabra – lassen Sie erneut LIST ausführen!
Verantwortlich für diese beeindruckende Mutation ist die Verwendung von eigentlichen Tokens im Quote Mode, wo sie zunächst nur Grafikzeichen ergeben; nach Löschen des Anführungszeichens geraten sie jedoch in den Normal Mode und werden zu Basic-Schlüsselwörtern (siehe »C 64 für Insider«).

Einzeiler #54: Das kürzeste Adventurespiel der Welt

Das Spiel heißt »Nosferatu«. Im Spiel selbst stehen Sie einem Vampir gegenüber und müssen herausfinden, wie Sie ihn besiegen. Erlaubt sind deutsche Zweiwortkommandos.

Listing 2.54: EZ ADVENTURE

```
1 PRINT"EIN VAMPIR!":INPUTA$:PRINT"SIEG!":IFA$<>"WIRF KNOBLAUCH"THENPRINT"(UP)KL
APPT NICHT!(DOWN)":GOTO1
2 REM *** NOSFERATU, DAS KUERZESTE
3 REM *** ADVENTURE DER WELT
```

Eine Alternative für die Lösung besteht darin, daß Sie anstatt »WIRF KNOBLAUCH« einen Text wie »ZEIGE KREUZ« eintippen. Sie können sich auch andere Situationen überlegen, was das Programm noch zum kürzesten Adventure-Generator macht. Und noch etwas: Mogeln Sie nicht! Es wäre schade, wenn Sie schon aus dem Listing die Lösung entnehmen würden.
Zur Funktionsweise, die durchaus ernsthaft ist: Unabhängig von Ihrer Eingabe wird »SIEG!« ausgegeben. War es die richtige Eingabe, so wird der THEN-Teil nicht ausgeführt und das Programm endet, während sich noch der Text »SIEG!« am Bildschirm befindet. Andernfalls wird der Cursor wieder zurückbewegt (so daß er auf dem »S« von »SIEG!« landet) und der Text »KLAPPT NICHT!« überschreibt gewissermaßen das »SIEG!«; daraufhin bewirkt GOTO 1 einen Rücksprung in den Eingabemodus.

Einzeiler #55: Schachuhr

Auch ohne aktuellen Anlaß wurde dieser Einzeiler schon aufgenommen: Bei der Schach-WM, welche während des Schreibens dieses Kapitels noch nicht entschieden war, verlor Titelverteidiger Kasparow eine Partie unter anderem deshalb, weil er vergaß, die Schachuhr zu betätigen und dadurch wertvolle Bedenkzeit verschenkte. Mit diesem C64-Programm wäre ihm, dem Werbeträger einer anderen bekannten Computerfirma, dieses verhängnisvolle Mißgeschick vielleicht nicht passiert:

Listing 2.55: EZ SCHACHUHR

```
2 S=1-S:TI$=A$(S):FORI=1TO1:GETI:A$(S)=TI$:PRINT"(CLR)    ?"A$(1),,A$(Ø):IFA$(S)
<E$THENNEXT:GOTO2
```

Nachdem dieses Programm zunächst eine Bildschirmseite Listing verbraucht hatte, wurde die Kürzung des Programms aufgenommen. Schließlich schrumpfte es sich dann zu vorliegendem Einzeiler gesund, der aus Ihrem Computer eine richtige Schachuhr macht. So wird er gestartet:

```
A$(0)="000000":A$(1)="000000":E$="000020":PRINT"<SHIFT CLR/HOME>"; GOTO 2
```

Zuerst sind die Anfangszeiten der beiden Spieler zu definieren, die wohl jeweils Null sein dürften – es sei denn, der eine Spieler gewährt seinem Kontrahenten einen großzügigen Vorsprung. Danach definiert man die Endzeit, ebenfalls im Format der Systemvariablen TI$, bei welcher

das Programm stoppen soll, und startet den Einzeiler über GOTO 2 – nicht über RUN, um die Stringvariablen nicht zu löschen. Auf Druck der Zahlen 1–9 läuft jeweils die Uhr des Gegenspielers weiter. Will man eine Pause machen, drückt man irgendeinen Buchstaben. Mit GOTO 2 kann das Programm dann fortgesetzt werden (ohne sich an dem SYNTAX ERROR zu stören), worauf die jeweils andere Uhr als die, die vor Abbruch lief, weitergezählt wird.

Zur Erklärung: S=1-S ändert S jeweils von 0 auf 1 und umgekehrt. Nun wird die Uhr TI$ weitergestellt. Danach folgt eine FOR-NEXT-Schleife, die nur verlassen werden kann, wenn I eine der Zahlen 1–9 annimmt (was nur bei Drücken irgendeiner Zifferntaste geschieht). In der Schleife wird die jeweilige Zeit der beiden Spieler ausgegeben. Außerdem wird untersucht, ob die Endzeit erreicht ist.

Einzeiler #56: Einzeiler zur Eingabe von Einzeilern

Gerade bei der Eingabe von »fülligen« Einzeilern stört man sich oft daran, daß maximal 80 Zeichen pro Eingabe möglich sind. Es lassen sich nach Start des folgenden Einzeilers Basic-Zeilen bis zu 88 Zeichen Länge eingeben:

Listing 2.56: EZ EZ-EDITOR

```
1 PRINT"(CLR)(DOWN)(DOWN)";:SYS59749:SYS59749:PRINT"(HOME)(DOWN)(DOWN)":PRINTTAB
(7)"↑":PRINT"(HOME)(DOWN)":SYS42112
```

```
1 PRINT"(CLR)(DOWN)(DOWN)";
  SYS59749
  SYS59749
  PRINT"(HOME)(DOWN)(DOWN)"
  PRINTTAB(7)"↑"
  PRINT"(HOME)(DOWN)"
  SYS42112
```

Man startet also den Einzeiler: Es wird dabei der Bildschirm gelöscht, der Cursor auf die dritte Bildschirmzeile gebracht, und in der sechsten Zeile in der achten Spalte ein Hochpfeil ausgedruckt. Nun gibt man eine Zeile, beginnend bei der Cursorposition, ein, bis der Pfeil auf das letzte Zeichen der Eingabezeile weist. Auf dem Bildschirm ist nun also eine zweizeilige Basic-Zeile zu sehen, sowie acht weitere Zeichen in der dritten Basic-Zeile. Dies ergibt insgesamt 2*40 + 8 = 88 Zeichen, welche auch nach einem <RETURN> angenommen werden. Warum steht der Cursor aber erst in der dritten Bildschirmzeile? Ganz einfach: Denn nun kann man in der ersten Bildschirmzeile ein LIST <zeilennummer> eingeben. Die gelistete Basic-Zeile paßt dann genau in den 88-Zeilenbereich hinein. Aber am besten probieren Sie den Einzeiler einfach einmal selbst aus!

Zum Programm selbst: SYS 59749 erzeugt die wichtige Fortsetzungszeile, welche der C64 selbständig nach 40 Eingabezeilen ausdruckt (Scrolling nach unten, nächste Zeile löschen), SYS 42112 bewirkt ein Basic-Ende ohne READY-Meldung. Wenn Sie diese beiden ROM-Routinen untersuchen (siehe »C64 für Insider«), dürfte Ihnen das Prinzip klar werden.

Editor #2: Mega-Einzeiler

Auf der Masterdiskette befindet sich unter dem Namen »EX-LINE $C000« (Extended Line Editor, also »erweiterter Zeileneditor«) ein Maschinenprogramm, welches die Eingabe superlanger Einzeiler ermöglicht; damit wurden die nun letzten vier Einzeiler eingegeben, die allesamt sensationelle Ergebnisse bringen.
Zurück zu EX-LINE. Mit diesem Programm ist es nun möglich, Basic-Zeilen mit einer Länge von bis zu 252 Zeichen einzugeben. Nachdem man es mit LOAD"EX-LINE",8,1 geladen hat, kann man es nach NEW mit SYS 49152 starten. Danach steht das Ausrufezeichen als neuer Befehl zur Verfügung. Wenn man diesen Befehl eingibt, erscheint der Cursor in der linken oberen Ecke des Bildschirms. Nun kann man eine Programmzeile oder Basic-Befehle im Direktmodus eingeben. Dabei ist zu beachten, daß nach <RETURN> nur die ersten 6,5 Bildschirmzeilen übernommen werden, und zwar in Unabhängigkeit von der Cursorposition. Gibt man nach dem Ausrufezeichen noch eine Zeilennummer an, wird vorher der Bildschirm gelöscht und die entsprechende Zeile so gelistet, daß man auch überlange Zeilen editieren kann.
Da sich das Programm nicht mit der beliebten Eingabehilfe »Checksummer« verträgt, errechnet es noch eine Prüfsumme, welche sich mit PRINT PEEK(2) auslesen läßt. Sie entsteht einfach durch Addition der ASC-Werte aller eingegebenen Zeichen. In der Speicherzelle 2 steht dann das Low-Byte der Summe.
Achtung: Gibt man die Befehle mit Abkürzungen ein, resultiert daraus eine andere Prüfsumme, als wenn man die Kommandos ausschreibt.

Zum Programm: Es ist komplett in Maschinensprache geschrieben und steht im Speicher ab $c000. Nach dem Start mit SYS49152 wird zuerst das Basic-ROM in den darunterliegenden RAM-Bereich kopiert und so verändert, daß der Eingabepuffer fortan nicht mehr bei $0200, sondern bei $c200 liegt. Dies ist nötig, da er jetzt viel mehr Platz benötigt, als das Betriebssystem bei $0200 für ihn vorsehen würde (siehe »C64 für Insider«). Schließlich wird der Vektor $0308/$0309 (siehe »C64 für Insider«) auf die neue Routine gelenkt. Diese prüft, ob das erste Zeichen ein Ausrufezeichen ist. Falls ja, so wird zunächst der Bildschirm gelöscht und die angegebene Zeile gelistet; wenn keine Zeile angegeben war, wird dieser Teil übersprungen. Nach erfolgter Eingabe wird das RAM bei $a000 aktiviert, der Text im oberen Teil des Bildschirms in den Puffer bei $c200 kopiert, in Interpretercode umgewandelt und ausgeführt. Das ROM wird bei der nächsten Eingabe wieder eingeschaltet, so daß der Eingabepuffer wieder regulär bei $0200 liegt. Die nun folgenden vier Einzeiler sind eine beeindruckende Demonstration für die Leistungsfähigkeit dieser Eingaberoutine.

Einzeiler #57: Soft-Flash

Dieser Mega-Einzeiler bewirkt, daß die rote LED am Diskettenlaufwerk scheinbar stufenlos ein- und ausgeschaltet wird.

Listing 2.57: EZ SOFT-FLASH

```
1 OPEN1,8,15:FORI=0TO44:READA:PRINT#1,"M-W"CHR$(I)CHR$(5)CHR$(1)CHR$(A):NEXT:PRI
NT#1,"UC":DATA169,254,170,32,21,5,202,224,1,208,248,32,21,5,232,224,255,208,248,
240,235,138,72,73,255,168,169,248,141,0,28,202,208,248,169,240,141,,28,136,208,2
48,104,170,96
```

In der FOR-NEXT-Schleife wird ein Maschinenprogramm mittels M-W (Memory-Write) in das Floppy-RAM ab $0500 geschrieben. Der Floppy-Befehl »UC« bewirkt dann, daß dieses gestartet wird. Das Maschinenprogramm schaltet die LED so schnell an und aus, daß dies für das Auge nicht sichtbar ist. Dabei ändert sich die Lage der Hell- und Dunkelphase, so daß es scheint, als ob die Lampe langsam hell und dunkel würde. Weil das Programm als Endlosschleife geschrieben ist, läßt sich das Laufwerk nicht mehr ansprechen. Zur weiteren Verwendung der Floppy muß man diese also aus- und wieder einschalten.

Einzeiler #58: Sound-Flash

Eine leichte Abänderung von Einzeiler #57 ergibt, daß das Laufwerk gleichzeitig ein Geräusch von sich gibt:

Listing 2.58: EZ SOUND-FLASH

```
1 OPEN1,8,15:FORI=0TO44:READA:PRINT#1,"M-W"CHR$(I)CHR$(5)CHR$(1)CHR$(A):NEXT:PRI
NT#1,"UC":DATA169,254,170,32,21,5,202,224,1,208,248,32,21,5,232,224,255,208,248,
240,235,138,72,73,255,168,169,248,141,0,28,202,208,248,169,240,141,,28,136,208,2
48,104,170,96
```

Achtung: Dies ist für das Laufwerk alles andere als schonend! Deshalb befindet sich der Einzeiler nicht auf Diskette, sondern die Datei »EZ SOUND-FLASH« ist leer. Der Grund liegt darin, daß Sie gar nicht erst in Versuchung geraten sollen, diesen Einzeiler mit der wertvollen Masterdiskette zu testen! Doch mit Hilfe des Editors »EX-LINE« können Sie EZ SOUND-FLASH eingeben; ändern Sie dazu gegenüber Listing 2.57 und 2.58 die Zahl 240 in der zweituntersten Zeile (zwischen 169 und 141) in 004 um. Die zu ändernde Stelle ist in Listing 2.58 hervorgehoben; Listing 2.58 stellt somit nicht die endgültige Fassung dar!

Einzeiler #59: Der Cursor, den die Profis lieben

Dieser Mega-Einzeiler verwandelt den Cursor in einen Strich, der unter den Zeichen blinkt, wie man es von größeren Computern (C128, Amiga) kennt. Diese Routine verleiht eigenen Programmen ein professionelles Aussehen.

Listing 2.59: EZ STRICH-CURSOR

```
1 W=56333:Q=53248:Z=4↑5:POKEQ+24,52:POKE56576,0:POKE648,204:POKEW,1:POKE1,3:FORI
=0TO999:POKE52224+I,PEEK(Z+I):NEXT:FORI=0TOZ:A=Q+I:B=A+2*Z:L=Z*((IAND7)=7):POKEA
,PEEK(A):POKEB,PEEK(B):POKEA+Z,PEEK(A-L):POKEB+Z,PEEK(B-L):NEXT:POKE1,7:POKEW,12
9


1 W=56333
  Q=53248
  Z=4↑5
  POKEQ+24,52
  POKE56576,0
  POKE648,204
  POKEW,1
  POKE1,3
  FORI=0TO999
  POKE52224+I,PEEK(Z+I)
  NEXT
  FORI=0TOZ
  A=Q+I
  B=A+2*Z
  L=Z*((IAND7)=7)
  POKEA,PEEK(A)
  POKEB,PEEK(B)
  POKEA+Z,PEEK(A-L)
  POKEB+Z,PEEK(B-L)
  NEXT
  POKE1,7
  POKEW,129
```

Die ersten drei POKE-Befehle bewirken, daß der Bildschirm nach $cc00 und der Zeichensatz ins RAM (!) nach $d000 verlegt wird. Dies ist notwendig, da kein Basic-Speicher verlorengehen soll und der VIC nur 16 Kbyte auf einmal adressieren kann. Bevor man nun den Originalzeichensatz sichtbar machen kann (POKE 1,3), wird noch der Interrupt mit POKE56333,1 ausgeschaltet. Die erste FOR-NEXT-Schleife kopiert den Zeichensatz ins RAM. Dabei werden die reversen Zeichen so verändert, daß nur die unterste Reihe revers erscheint, also ein Unterstrich entsteht. Die folgenden POKEs bewirken schließlich, daß der Interrupt eingeschaltet wird und der I/O-Bereich bei $d000 wieder ansprechbar ist.

Einzeiler #60: Orientierungsprobleme...

Der verblüffendste Einzeiler zum Schluß:

Listing 2.60: EZ UPSIDE-DOWN

```
1 PRINT"(HOME)(RVS)  (OFF)-(RVS)9(OFF)‾D(RVS)Y(OFF),A(RVS)H(OFF)P(RVS)‾(OFF)▔(RVS
))(OFF)3(RVS)E(OFF)A(RVS))(OFF)@(RVS)E‾)┐E▔ (OFF)@(RVS)1‾.9#(OFF) ╲&B(RVS) ╲┐└┘X(OF
F)╲G(RVS)(%(OFF)B(RVS)Q‾X(OFF)╲G(RVS)( │‾▒▒▔%▔╲ ┐╥)(OFF)7(RVS)E(OFF)A└,A▔(RVS))DM
(OFF)X(RVS)┐)(OFF)@(RVS)M(OFF)@(RVS) │)(OFF)‾(RVS)M(OFF)TC(RVS))(OFF)A(RVS)M(OFF)
UC♠ (RVS))(OFF)@(RVS)*(H=(OFF)@D(RVS)Y▒▬=(OFF)@E(RVS)Y▒▌=(OFF)@F(RVS)Y▒ =(OFF)@G
(RVS)Y▒◥H▒┐└(OFF)└1(RVS) │(OFF)":SYS1024
```

```
1 PRINT"(HOME)(RVS)  (OFF)-(RVS)9(OFF)‾D(RVS)Y(OFF),A(RVS)H(OFF)P(RVS)‾(OFF)▔(RVS
))(OFF)3(RVS)E(OFF)A(RVS))(OFF)@(RVS)E‾)┐E▔ (OFF)@(RVS)1‾.9#(OFF) ╲&B(RVS) ╲┐└┘X(OF
F)╲G(RVS)(%(OFF)B(RVS)Q‾X(OFF)╲G(RVS)( │‾▒▒▔%▔╲ ┐╥)(OFF)7(RVS)E(OFF)A└,A▔(RVS))DM
(OFF)X(RVS)┐)(OFF)@(RVS)M(OFF)@(RVS) │)(OFF)‾(RVS)M(OFF)TC(RVS))(OFF)A(RVS)M(OFF)
UC♠ (RVS))(OFF)@(RVS)*(H=(OFF)@D(RVS)Y▒▬=(OFF)@E(RVS)Y▒▌=(OFF)@F(RVS)Y▒ =(OFF)@G
(RVS)Y▒◥H▒┐└(OFF)└1(RVS) │(OFF)"
  SYS1024
```

Dieses Programm stellt den ganzen Bildschirm »auf den Kopf«. Dies bezieht sich auf alle (!) Ein- und Ausgaben. Auch Zeichen, die mit POKE auf den Bildschirm kommen, erscheinen an der entsprechend entgegengesetzten Stelle auf dem Kopf. Die meisten Programme vertragen sich damit sogar gut, weil nur der IRQ-Vektor verändert wird. Nach Reset oder <RUN/STOP RESTORE> kann das Programm mit SYS300 reaktiviert werden.
Programmbeschreibung: Das Maschinenprogramm wird mittels PRINT auf den Bildschirm gebracht und kopiert zunächst eine IRQ-Routine in den Stack ab Adresse 300; dieser Bereich ist dafür prädestiniert, weil er nach einem Reset nicht gelöscht wird und sich das Programm dort nicht stört. Dann wird der gesamte Zeichensatz auf den Kopf gestellt und ins RAM bei \$d000 kopiert. Der VIC wird veranlaßt, den Zeichensatz aus \$d000 und den Bildschirm aus \$e000 zu lesen, und schließlich wird die Interruptroutine aktiviert, welche 60mal in der Sekunde den Bildschirm in umgekehrtem Zustand nach \$e000 kopiert.
Insgesamt wird also der sichtbare Bildschirm genau um 180 Grad gedreht.

2.2 Mehrzeiler

Auch mit wenigen Zeilen – es muß ja nicht nur eine einzige sein – lassen sich interessante Effekte erzielen.

2.2.1 Master Mind als Vierzeiler

Listing 2.61: 4Z MASTER MIND

```
1 INPUT"(CLR)(DOWN)STELLEN";S
  E=INT(10↑S*RND(0))
  GOSUB2
  FORI=1TOS
  L(I)=E(I)
  NEXT
  GOTO3
2 FORI=1TOS
  E(I)=E-10*INT(E/10)
  E=INT(E/10)
  C(I)=0
  B(I)=0
  R=R-(L(I)=E(I))
  NEXT
  RETURN
3 V=V+1
  INPUTE
  R=0
  F=0
  GOSUB2
  FORI=1TOS
  FORJ=1TOS
  B=(L(J)=E(I))ANDNOTB(I)ANDNOTC(J)
4 B(I)=B(I)+B
  C(J)=C(J)+B
  F=F-B
  NEXTJ,I
  PRINT"(UP)"TAB(16)R"   "F-R"   "V
  IFR<STHEN3
```

```
1 INPUT"(CLR)(DOWN)STELLEN";S:E=INT(10↑S*RND(0)):GOSUB2:FORI=1TOS:L(I)=E(I):NEXT
:GOTO3
2 FORI=1TOS:E(I)=E-10*INT(E/10):E=INT(E/10):C(I)=0:B(I)=0:R=R-(L(I)=E(I)):NEXT:R
ETURN
3 V=V+1:INPUTE:R=0:F=0:GOSUB2:FORI=1TOS:FORJ=1TOS:B=(L(J)=E(I))ANDNOTB(I)ANDNOTC
(J)
4 B(I)=B(I)+B:C(J)=C(J)+B:F=F-B:NEXTJ,I:PRINT"(UP)"TAB(16)R"   "F-R"   "V:IFR<STHE
N3
```

Dieses Programm stammt von Hans Haberl, dem Autor so berühmt gewordener Programme wie Hi-Eddi, Pagefox und Printfox; es entstand als Nebenprodukt seiner Einzeilerbemühungen. Bei Master Mind geht es darum, eine Zahl, die sich der Computer »ausdenkt«, zu erraten. Am Anfang gibt man die Anzahl der Stellen der zu erratenden Zahl ein; sie darf maximal acht sein (man hat aber schon mit drei oder vier genug zu knobeln!). In der erste Spalte muß man nun jeweils eine Zahl eingeben, der Computer zeigt in den folgenden drei Spalten an:

1. Anzahl der richtigen Ziffern an der richtigen Stelle
2. Anzahl der richtigen Ziffern an der falschen Stelle
3. Anzahl der Versuche

Beispiel eines Spiels (nicht nachvollziehbar wegen des Zufallsgenerators, der immer wieder neue Ergebnisse liefert):

Stellen? 4	
? 1123	1 0 1
? 4456	0 0 2
? 7789	2 1 3
? 8989	1 0 4
? 7979	1 1 5
? 7187	4 0 6

Es geht natürlich darum, die Zahl mit möglichst wenigen Versuchen zu erraten. Hier die Variablenliste:

S	Anzahl der Stellen
E,E()	Eingabe, Ziffern der Eingabe
L()	Ziffern der Lösung
B(),C()	Belegungsvektoren für E und L
R	Richtige Ziffern an richtiger Stelle
F	Alle richtigen Ziffern
V	Versuche
B,I,J	Hilfs- und Schleifenvariablen

2.2.2 Geräusche

Hier sei noch ein Grundgerüst für eigene Geräuscheffekte gegeben, das auch Nicht-Musikfreaks großen Nutzen bringt:

Listing 2.62: SOUND-BEISPIEL

```
10 S=54272
20 FORL=0TO24
   POKES+L,0
   NEXT
30 POKES+O,0
   POKES+1,18
40 POKES+5,1*16+11
50 POKES+22,110
60 POKES+23,15*16+3
70 POKES+24,5*16+15
80 POKES+4,0
   POKES+4,129
```

```
90 FORJ=1TO255
   POKES+O,J
   NEXT
100 FORA=1TO1000
    NEXT
    GOTO80
```

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+O,0:POKES+1,18
40 POKES+5,1*16+11
50 POKES+22,110
60 POKES+23,15*16+3
70 POKES+24,5*16+15
80 POKES+4,0:POKES+4,129
90 FORJ=1TO255:POKES+O,J:NEXT
100 FORA=1TO1000:NEXT:GOTO80
```

Durch Veränderung von Filterfrequenz und durch verschiedene Filter entstehen einfache Geräusche. Das Beispiel erzeugt einen Schuß.

Folgende Bedeutung haben die einzelnen Zeilen:

10	Definition des Basisregisters (konstant)	60	Resonanz
20	Lösch-Schleife aller Register (konstant)	70	Paß
30	Frequenz	80	Wellenform
40	Hall	90	Schleife und POKE
50	Grenzfrequenz	100	Warteschleife und GOTO

2.2.3 Reformat als Dreizeiler

Bei der Formatierung ohne ID werden nur die BAM und der Block 18/1 gelöscht. 18/1 weist im Normalfall immer auf 18/4, wo das Directory fortgesetzt wird. Wenn man auf die ersten acht Einträge verzichten wollte, könnte man zumindest den Rest (bei 144 Einträgen immerhin 136) durch drei Zeilen retten:

Listing 2.63: 3Z REFORMAT

```
1 PRINT"DIESES PROGRAMM IST ZWAR INTERESSANT,(DOWN)"
2 PRINT"WAS DIE FUNKTIONSWEISE ANBELANGT, SOLLTE"
3 PRINT"ABER NICHT VERWENDET WERDEN. VIEL BESSER"
4 PRINT"IST DAS UTILITY >>DISC-WIZARD<<."
5 PRINT"(DOWN)(DOWN)(DOWN)(RVS)UNVERBESSERLICHE MUESSEN >>GOTO 10<<     ":
6 PRINT"EINGEBEN."
9 STOP
```

```
10 OPEN1,8,15
   OPEN2,8,2,"#"
   PRINT#1,"U1 2 0 18 1"
20 PRINT#1,"M-W"CHR$(0)CHR$(5)CHR$(2)CHR$(18)CHR$(4)
30 PRINT#1,"U2 2 0 18 1"
   PRINT#1,"V"
   CLOSE2
   CLOSE1
```

```
1 PRINT"DIESES PROGRAMM IST ZWAR INTERESSANT,(DOWN)"
2 PRINT"WAS DIE FUNKTIONSWEISE ANBELANGT, SOLLTE"
3 PRINT"ABER NICHT VERWENDET WERDEN. VIEL BESSER"
4 PRINT"IST DAS UTILITY >>DISC-WIZARD<<."
5 PRINT"(DOWN)(DOWN)(DOWN)(RVS)UNVERBESSERLICHE MUESSEN >>GOTO 10<<     ";
6 PRINT"EINGEBEN."
9 STOP
10 OPEN1,8,15:OPEN2,8,2,"#":PRINT#1,"U1 2 0 18 1"
20 PRINT#1,"M-W"CHR$(0)CHR$(5)CHR$(2)CHR$(18)CHR$(4)
30 PRINT#1,"U2 2 0 18 1":PRINT#1,"V":CLOSE2:CLOSE1
```

Das Programm ändert die Zeiger des ersten Directory-Blocks. Danach sind wieder, bis auf die ersten acht, alle Programme vorhanden. Schreibzugriffe könnten jedoch die Diskette zerstören, weshalb noch die Diskette validiert wird. Beim Validieren werden die Programme als belegt gekennzeichnet und somit vor Überschreiben geschützt.

Achtung: Dieses Programm ist nur als Anregung, nicht jedoch zum tatsächlichen Gebrauch bestimmt!

2.2.4 Basic-Erweiterungen durchschaut

In den meisten Basic-Dialekten und Basic-Erweiterungen werden die Schlüsselworte als Tokens (siehe »C64 für Insider«) gespeichert. Folgendes Programm erzeugt eine vollständige Liste aller Tokens und der zugehörigen Befehle, wenn Sie es unter dem zu untersuchenden Dialekt laden und starten:

Listing 2.64: TOKEN-FINDER

```
400 POKE2,127
410 Z=PEEK(2)+1
420 IFZ=256THEN550
430 H=INT(Z/16)
440 L=Z-16*H
450 IFH>9THENH=H+7
460 IFL>9THENL=L+7
```

```
470 H$=CHR$(48+H)
480 L$=CHR$(48+L)
490 PRINTCHR$(147);Z;"...";H$;L$;"...*"
500 PRINT"RUN 410"
510 POKE2,Z
520 POKE631,19
    POKE632,13
530 POKE633,13
    POKE198,3
540 END
550 A=PEEK(44)*256+PEEK(43)+12
560 FORI=0TO127
570 POKEA+14*I,I+128
580 NEXTI
590 POKE2,29
    PRINTCHR$(147)
600 Z=PEEK(2)+1
    PRINTCHR$(19)10*Z
    PRINT"R/600"
    POKE631,19
    POKE632,13
    POKE633,13
    POKE198,3
    POKE2,Z
    END
```

```
400 POKE2,127
410 Z=PEEK(2)+1
420 IFZ=256THEN550
430 H=INT(Z/16)
440 L=Z-16*H
450 IFH>9THENH=H+7
460 IFL>9THENL=L+7
470 H$=CHR$(48+H)
480 L$=CHR$(48+L)
490 PRINTCHR$(147);Z;"...";H$;L$;"...*"
500 PRINT"RUN 410"
510 POKE2,Z
520 POKE631,19:POKE632,13
530 POKE633,13:POKE198,3
540 END
550 A=PEEK(44)*256+PEEK(43)+12
560 FORI=0TO127
570 POKEA+14*I,I+128
580 NEXTI
590 POKE2,29:PRINTCHR$(147)
600 Z=PEEK(2)+1:PRINTCHR$(19)10*Z:PRINT"R/600":POKE631,19:POKE632,13:POKE633,13:
POKE198,3:POKE2,Z:END
```

Die Werte der Tokens liegen im Bereich $80–$FF (#128–#255). Das Programm erzeugt daher zunächst Zeilen mit diesen Nummern und dem jeweiligen Byte und poket dann die Tokens in die Zeile hinein. Am Schluß löscht sich das Hauptprogramm selbst, und im Programmspeicher verbleibt die Liste der Tokens und ihre Bedeutung, welche nun über LIST abgerufen

werden kann. In manchen Basic-Dialekten steigt der Computer beim Code 204 ($CC) aus (siehe »C64 für Insider«); durch Löschen der Zeile 204 kann dies umgangen werden. Interessant ist übrigens die Auto-Löschroutine in Zeile 600, welche als Einzeiler ihren eigenen Wert hätte. Es sei auf Kapitel 1.4 hingewiesen, wo die programmtechnischen Grundlagen erörtert werden.

2.2.5 Sprite-Editor als Zweizeiler

Bestimmt gibt es schon, ohne Übertreibung, weiter über hundert Sprite-Editoren, die für den C64 im Umlauf sind – sei es als eigenständige Programme oder als Funktionsteile eines größeren Systems. Jedes neue Produkt dieser Art beansprucht für sich den großartigsten Leistungsumfang, noch nie dagewesene Funktionen und den höchsten Eingabekomfort. Doch ein rudimentärer Sprite-Editor läßt sich in sage und schreibe zwei Basic-Zeilen unterbringen:

Listing 2.65: 2Z SPRITE-EDITOR

```
1 FORI=ØTO2
  A=Ø
  FORN=ØTO7
  A=A-2↑(7-N)*(PEEK(1024+N+I*8)=42)
  NEXT
2 PRINTA;
  NEXT
  PRINT
  X=X+1
  IFX<21THEN1
```

```
1 FORI=ØTO2:A=Ø:FORN=ØTO7:A=A-2↑(7-N)*(PEEK(1024+N+I*8)=42):NEXT
2 PRINTA;:NEXT:PRINT:X=X+1:IFX<21THEN1
```

Und so entwirft man ein Sprite: Im Direktmodus wird das Programm geladen (aber noch nicht gestartet) und der Bildschirm mit <SHIFT CLR/HOME> gelöscht. Jetzt kann, angefangen in der linken oberen Bildschirmecke, das Sprite entworfen werden:

<*>	setzt einen Punkt
<SPACE>	löscht bzw. überspringt einen Punkt

Der Cursor wird, wie gewohnt, mit den Cursortasten bewegt. Das Eingabefeld hat eine Größe von 24 Spalten und 21 Zeilen. Sind Sie schließlich mit Ihrem Kunstwerk zufrieden und wollen die Sprite-Daten ausgeben, ist der Cursor am Anfang der vorletzten (unbedingt beachten!) Bildschirmzeile zu positionieren. RUN <RETURN> an dieser Stelle startet schließlich den

Sprite-Editor, der die Daten errechnet und auf den Bildschirm ausgibt. Der Übernahme in eigene Programme steht nichts mehr im Wege.
Zur Funktionsweise: Es wird jeweils die oberste Bildschirmzeile in 3 DATA-Werte umgewandelt und der Bildschirm bei der Ausgabe der Daten um eine Zeile nach oben gescrollt (deshalb die Positionierung in der vorletzten Bildschirmzeile!); durch das Scrolling wiederum wird die Schleife erneut durchlaufen, so daß die jetzige oberste Zeile bereits die nächsten DATA-Werte enthält.
Der einzige Nachteil daran ist, daß das erstellte Sprite während der Datenberechnung vom Bildschirm verschwindet; Korrekturen lassen sich also nicht mehr durchführen. Die beiden folgenden Zeilen stellen denselben Editor wie Listing 2.65 dar, jedoch mit Ausnahme des genannten Nachteils:

Listing 2.66: SPRITE-EDITOR V2

```
1 FORI=ØTO2
  A=Ø
  FORN=ØTO7
  A=A-2↑(7-N)*(PEEK(1Ø24+4Ø*X+N+I*8)=42)
  NEXT
  A(I+1)=A
2 NEXT
  PRINTTAB(25)A(1)A(2)A(3)
  X=X+1
  IFX<21THEN1



1 FORI=ØTO2:A=Ø:FORN=ØTO7:A=A-2↑(7-N)*(PEEK(1Ø24+4Ø*X+N+I*8)=42):NEXT:A(I+1)=A
2 NEXT:PRINTTAB(25)A(1)A(2)A(3):X=X+1:IFX<21THEN1
```

Die errechneten Sprite-Werte werden rechts vom Sprite ausgegeben; der TAB-Befehl in der zweiten Programmzeile dient zum Überspringen der vorausgehenden Bildschirminhalte. Die Berechnung der PEEK-Adresse wird dadurch etwas komplizierter, daß jetzt die Nummer der aktuellen Zeile (0–21) durch Addition von »40*X« berücksichtigt wird.

2.2.6 Mini-Monitor als Dreizeiler

Und noch ein Kürze-Rekord wird aufgestellt: »Minmon« ist wohl der kürzeste Speichermonitor nach dem Einzeiler DI-AS (siehe 2.1).

Listing 2.67: 3Z MINMON

```
20 INPUT"STARTADRESSE";SA
40 PRINTSA
   FORX=ØTO3
   PE=PEEK(SA+X)
   PA=PE
   IFPE<32ORPE<16ØANDPE>127THENPE=46
80 PRINT"(UP)"TAB(6+X*4)PATAB(25+X)CHR$(PE)
   NEXT
   SA=SA+4
   GOTO4Ø
```

```
2Ø INPUT"STARTADRESSE";SA
4Ø PRINTSA:FORX=ØTO3:PE=PEEK(SA+X):PA=PE:IFPE<32ORPE<16ØANDPE>127THENPE=46
8Ø PRINT"(UP)"TAB(6+X*4)PATAB(25+X)CHR$(PE):NEXT:SA=SA+4:GOTO4Ø
```

Die Speicherstellen werden in Zeile 40 ausgelesen und geprüft, ob sie als ASCII-Zeichen darstellbar sind. In Zeile 80 werden diese Werte (jeweils 4) formatiert ausgegeben (links die Adresse, in der Mitte der Dezimalwert, rechts der ASCII-Wert). Probieren Sie zum Beispiel als Startadresse 2048 oder 41117.

2.2.7 Wenn Pythagoras einen C64 gehabt hätte...

...dann hätte er seine »pythagoreischen Zahlentripel« möglicherweise von folgendem Zweizeiler berechnen lassen:

Listing 2.68: 2Z PYTHAGORAS

```
1Ø FORX=1TO8
   FORY=1TO8
   A=X*X-Y*Y
   B=2*X*Y
   C=X*X+Y*Y
   IFINT(A)>=1THENPRINTINT(A),B,INT(C)
2Ø NEXTY,X
```

```
1Ø FORX=1TO8:FORY=1TO8:A=X*X-Y*Y:B=2*X*Y:C=X*X+Y*Y:IFINT(A)>=1THENPRINTINT(A),B,
INT(C)
2Ø NEXTY,X
```

Dieses Programm gibt alle diejenigen Zahlen a, b und c aus, für welche der Satz des Pythagoras (a↑2 + b↑2 = c↑2) erfüllt ist. Aus Zeitgründen wird nicht potenziert, sondern multipliziert. Zur Veränderung des Bereiches sind die Schleifenbereiche für X und Y zu ändern.

2.2.8 Floppy-Status abfragen

Folgende zwei Zeilen lesen den Status des Floppy-Laufwerkes ein und geben ihn auf den Bildschirm aus:

Listing 2.69: 2Z FLOPPY-STATUS

```
10 OPEN1,8,15
   FORX=1TO40
   POKE781,1
   SYS65478
   SYS65487
   SYS65490
   SYS65484
   IFST=0THENNEXT
20 CLOSE1
```

```
10 OPEN1,8,15:FORX=1TO40:POKE781,1:SYS65478:SYS65487:SYS65490:SYS65484:IFST=0THE
NNEXT
20 CLOSE1
```

Vielleicht sagen Sie jetzt zu Recht: Das geht doch viel einfacher mit

```
10 OPEN 1,8,15:INPUT#1,A,A$,B,C:?A,A$,B,C:CLOSE 1
```

Richtig, doch der einfachste Weg ist nicht immer der beste; bei Listing 2.69 können Sie nämlich auch die Zeilennummern weglassen und die beiden Zeilen hintereinander eingeben, um den Floppy-Status selbst im Direktmodus auszulesen! Für diejenigen, die kein entsprechendes Floppy-Tool (oder Paradoxon Basic, s. Kapitel 5) haben, ist dies eine hilfreiche Ersatzlösung. Durch die Verwendung von Betriebssystemroutinen (siehe »C64 für Insider«) wird der INPUT#-Befehl, welcher nur im Programm-Modus funktioniert, mit Erfolg umgangen. Ein weiterer Vorteil: Der beschriebene Algorithmus ist schnell in Assemblersprache übersetzt.

2.2.9 Disketten-Namen auslesen

Nach dem in 2.2.8 vorgestellten Verfahren ist auch der Name einer Diskette in nur zwei Basic-Zeilen, welche ebenfalls im Direktmodus funktionieren, darstellbar (siehe Seite 118):

Listing 2.70: 2Z DISK-NAME

```
10 OPEN1,8,15
   OPEN2,8,2,"#"
   PRINT#1,"U1 2 0 18 0"
   PRINT#1,"B-P 2 144"
20 FORX=1TO16
   POKE781,2
   SYS65478
   SYS65487
   SYS65490
   SYS65484
   NEXT
   CLOSE2
   CLOSE1
```

```
10 OPEN1,8,15:OPEN2,8,2,"#":PRINT#1,"U1 2 0 18 0":PRINT#1,"B-P 2 144"
20 FORX=1TO16:POKE781,2:SYS65478:SYS65487:SYS65490:SYS65484:NEXT:CLOSE2:CLOSE1
```

Will man nämlich ein Programm abspeichern und möchte dazu den Namen einer eingelegten Diskette wissen, leistet der Zweizeiler als Direkteingabe einige Hilfe.

2.2.10 Primzahlen als Vierzeiler

Auch als Einzeiler #34 habe ich ein Programm zur Berechnung von Primzahlen vorgestellt. Folgender Vierzeiler ist jedoch um einiges leistungsfähiger und komfortabler:

Listing 2.71: 4Z PRIMZAHLEN

```
10 INPUT"(CLR)PRIMZAHLEN BIS";H
   Z=3
   T=INT(SQR(H)+1)
   W=(T-1)/2
   G=T*T
   DIMP%(G)
20 FORX=1TOW
   FORY=Z*ZTOGSTEPZ*2
   P%(Y)=1
   NEXT
   Z=Z+2
   NEXT
30 PRINT2;
   FORX=3TOHSTEP2
   IFP%(X)=.THENPRINTX;
40 NEXT
```

```
10 INPUT"(CLR)PRIMZAHLEN BIS";H:Z=3:T=INT(SQR(H)+1):W=(T-1)/2:G=T*T:DIMP%(G)
20 FORX=1TOW:FORY=Z*ZTOGSTEPZ*2:P%(Y)=1:NEXT:Z=Z+2:NEXT
30 PRINT2;:FORX=3TOHSTEP2:IFP%(X)=.THENPRINTX;
40 NEXT
```

Er arbeitet ebenfalls nach dem Sieb des Eratosthenes, auch unter dem englischen Namen »Sieve« bekannt geworden. Das Programm benötigt für die Primzahlen bis 1000 weniger als 9 Sekunden; da mit Feldvariablen gearbeitet wird, können nun Primzahlen bis 19300 ausgerechnet werden. Damit handelt es sich sowohl um eines der schnellsten Basic-2.0-Programme seiner Art als auch um eines der flexibelsten, was die Größe des Zahlenbereiches betrifft.

2.2.11 Frogger-Variante in 23 Zeilen

Basierend auf dem Einzeiler #22 ist es gelungen, ein grafisches Froggerspiel in 23 Basic-Zeilen zu realisieren. Das Ziel ist, den Frosch wohlbehütet auf die andere Seite der Straße zu führen. Berührung mit Fremdkörpern führt zu Abzug eines Froschlebens.

Listing 2.72: 23Z FROSCH

```
0 V=53248
  POKE2042,13
  POKE2043,14
1 PRINT"(CLR)@@@?(RVS)  (OFF)@▚(RVS)C—▞▔(OFF)@(RVS) ┌(OFF)←(RVS)— ╵"(OFF)@(RVS)▚▞▚(O
FF)@▚(RVS)▚@▚▂—▚▂   (OFF)?(RVS)  ▚▂ ▚▂—(OFF)▚(RVS)▚@▚▞▚(OFF)@(RVS) ┌▔(OFF)@(RVS) ┌";
2 PRINT"←(RVS)—▞▔@▚(RVS)C—(OFF)←(RVS)  (OFF)@@@@(RVS)▚(OFF)@(RVS)▘ (OFF)@G(RVS)▚ (
OFF)?(RVS)▚▔(OFF)▚(RVS)▚▔▚▞▚▔▚▞▚ ▔▚▞▘ (OFF)▚(RVS)▚▞▚(OFF)?(RVS)▚▞▚(OFF)←(RVS)▚▞▚(OFF)O"
;
3 PRINT"(RVS)▚▞▚(OFF)O(RVS)▚▞ (OFF)O(RVS)▚▔(OFF)←(RVS)▚▔(OFF)←(RVS)▚"(OFF)?(RVS)▚▞
(OFF)?(RVS)▚▞▚(OFF)?(RVS)▚▞ (OFF)←(RVS)▚▔(OFF)G(RVS)▚—(OFF)A ▘@"
4 FORT=0TO126
  POKE832+T,PEEK(1024+T)
  NEXT
  POKEV+41,13
  POKEV+42,6
  POKEV+27,12
5 POKE650,128
  POKE649,2
  POKEV+32,0
  POKEV+33,5
  POKEV+23,8
  POKEV+29,8
  S=300
  G=0
6 POKEV+16,8
  POKEV+6,30
  POKEV+7,120
  Y=100
  X=40
  POKEV+4,X
  POKEV+5,Y
  POKEV+21,12
7 POKEV+30,PEEK(V+30)AND251
  POKEV+31,PEEK(V+31)AND243
8 X$="(RVS)  (DOWN)(LEFT)  (DOWN)(LEFT)  (DOWN)(DOWN)(DOWN)(DOWN)(LEFT)  (DOWN)(LEFT
)  (DOWN)(LEFT)  (DOWN)(DOWN)(DOWN)(DOWN)(LEFT)  (DOWN)(LEFT)  (DOWN)(LEFT)  (DOWN)(D
OWN)(DOWN)(DOWN)(LEFT)  (DOWN)(LEFT)  (DOWN)(LEFT)  "
9 PRINT"(CLR)(RED)";TAB(10);X$
  PRINT"(HOME)(BLK)(DOWN)(DOWN)";TAB(17);X$
  PRINT"(HOME)(WHT)";TAB(24);X$
  PRINT"(HOME)(GRY2)(DOWN)(DOWN)";TAB(31);X$
10 POKE56322,224
   J=PEEK(56321)
   IF(JAND1)=0THENY=Y-15
   GOTO17
```

```
11 IFPEEK(V+31)AND4=4THENPRINT"(BLK)(CLR)TOT";G;"FROESCHE GERETTET"
   WAIT145,16,16
   GOTO5
12 IFPEEK(V+30)=12THENPRINT"(CLR)(BLK)GERETTET !"
   S=S-50
   G=G+1
   WAIT145,16,16
   GOTO6
13 IF(JAND2)=0THENY=Y+15
   GOTO17
14 IF(JAND4)=0THENX=X-15
   GOTO17
15 IF(JAND8)=0THENX=X+15
   GOTO17
16 POKE172,PEEK(60656)
   POKE780,PEEK(216)
   SYS59848
   PRINT
   FORL=1TOS
   NEXT
   GOTO10
17 IFY>230THENY=230
18 IFY< 40THENY= 40
19 IFX>63AND(PEEK(V+16)AND4)=4THENX=63
20 IFX<25AND(PEEK(V+16)AND4)<>4THENX=25
21 IFX<0AND(PEEK(V+16)AND4)=4THENPOKEV+16,8
   X=255+X
22 IFX>255THENPOKEV+16,PEEK(V+16)OR4
   X=X-255
23 POKEV+4,X
   POKEV+5,Y
   GOTO10
```

```
0 V=53248:POKE2042,13:POKE2043,14
1 PRINT"(CLR)@@@?(RVS)  (OFF)@▚(RVS)C−▚▀(OFF)@(RVS) ┌(OFF)←(RVS)− |▘(OFF)@(RVS)▚▚(O
FF)@▚(RVS)▚@▚▄−▚▄   (OFF)?(RVS) ▚▄ ▚▄−(OFF)▚(RVS)▚@▚▚(OFF)@(RVS) ┌▀(OFF)@(RVS)┌";
2 PRINT"←(RVS)−▚▀@▚(RVS)C−(OFF)←(RVS)  (OFF)@@@@(RVS)▚(OFF)@(RVS)▘ (OFF)@G(RVS)▚ (
OFF)?(RVS)▚▀(OFF)▚(RVS)▚▀▚▚▀▚▚ ▀▚▚ (OFF)▚(RVS)▚▚(OFF)?(RVS)▚▚(OFF)←(RVS)▚▚(OFF)O"
;
3 PRINT"(RVS)▚▚(OFF)O(RVS)▚ (OFF)O(RVS)▚▀(OFF)←(RVS)▚▀(OFF)←(RVS)▚▘(OFF)?(RVS)▚
(OFF)?(RVS)▚▚(OFF)?(RVS)▚ (OFF)←(RVS)▚▀(OFF)G(RVS)▚−(OFF)A▘@"
4 FORT=0TO126:POKE832+T,PEEK(1024+T):NEXT:POKEV+41,13:POKEV+42,6:POKEV+27,12
5 POKE650,128:POKE649,2:POKEV+32,0:POKEV+33,5:POKEV+23,8:POKEV+29,8:S=300:G=0
6 POKEV+16,8:POKEV+6,30:POKEV+7,120:Y=100:X=40:POKEV+4,X:POKEV+5,Y:POKEV+21,12
7 POKEV+30,PEEK(V+30)AND251:POKEV+31,PEEK(V+31)AND243
8 X$="(RVS)  (DOWN)(LEFT)  (DOWN)(LEFT)  (DOWN)(DOWN)(DOWN)(DOWN)(LEFT)  (DOWN)(LEFT
)  (DOWN)(LEFT)  (DOWN)(DOWN)(DOWN)(DOWN)(LEFT)  (DOWN)(LEFT)  (DOWN)(LEFT)  (DOWN)(D
OWN)(DOWN)(DOWN)(LEFT)  (DOWN)(LEFT)  (DOWN)(LEFT)  "
9 PRINT"(CLR)(RED)";TAB(10);X$:PRINT"(HOME)(BLK)(DOWN)(DOWN)";TAB(17);X$:PRINT"(
HOME)(WHT)";TAB(24);X$:PRINT"(HOME)(GRY2)(DOWN)(DOWN)";TAB(31);X$
10 POKE56322,224:J=PEEK(56321):IF(JAND1)=0THENY=Y-15:GOTO17
11 IFPEEK(V+31)AND4=4THENPRINT"(BLK)(CLR)TOT";G;"FROESCHE GERETTET":WAIT145,16,1
6:GOTO5
12 IFPEEK(V+30)=12THENPRINT"(CLR)(BLK)GERETTET !":S=S-50:G=G+1:WAIT145,16,16:GOT
O6
13 IF(JAND2)=0THENY=Y+15:GOTO17
14 IF(JAND4)=0THENX=X-15:GOTO17
15 IF(JAND8)=0THENX=X+15:GOTO17
16 POKE172,PEEK(60656):POKE780,PEEK(216):SYS59848:PRINT:FORL=1TOS:NEXT:GOTO10
```

```
17 IFY>23ØTHENY=23Ø
18 IFY< 4ØTHENY= 4Ø
19 IFX>63AND(PEEK(V+16)AND4)=4THENX=63
2Ø IFX<25AND(PEEK(V+16)AND4)<>4THENX=25
21 IFX<ØAND(PEEK(V+16)AND4)=4THENPOKEV+16,8:X=255+X
22 IFX>255THENPOKEV+16,PEEK(V+16)OR4:X=X-255
23 POKEV+4,X:POKEV+5,Y:GOTO1Ø
```

Für die Kürze des Programms ist der Spielwert geradezu sensationell.

2.2.12 Ein Ausflug in die Urzeit des Telespiels...

...erwartet Sie mit folgendem Spielprogramm, dessen Listing am Bildschirm gerade eine Seite (!) beansprucht, also vollständig auf einmal am Bildschirm zu sehen ist:

Listing 2.73: SCREEN-PONG 64

```
Ø PRINT"(CLR)(RVS)(YEL)";
  FORI=ØTO39
  PRINT" ";
  NEXT
  V=49152
  FORJ=1TO7
  READA$
  FORI=1TO36
1 POKEV,1Ø*(ASC(MID$(A$,I*2-1,1))-65)+VAL(MID$(A$,I*2,1))
  V=V+1
  NEXT
  NEXT
2 V=832
  FORI=VTO96Ø
  POKEI,Ø
  NEXT
  FORI=ØTO2Ø
  POKEV+I*3,255
  NEXT
  V=53248
  POKE2,Ø
8 POKEV+16,2
  POKEV+4,Ø
  POKEV+21,7
  Q=(PEEK(2)AND3)
  P(Q)=P(Q)+1
  PRINT"(HOME)(RVS)"P(2);
9 PRINTSPC(25);P(1)
  SYS49152
  GOTO8
5Ø DATAQ9O3O1A1U8O1A3U8Q9AØO1D3U8O1C3U8O1C9U8Q9H2O1A2U8Q9D2O1AØU8Q2B3O2Y9A7O2Y8
51 DATAA7X2O2ZØA7X2O2N1A3O2N4A3Q2A4O2EØU8X2O2D9U8X2O2M8A3O2N7A3X2O2D2U8Q9R3O1A2
52 DATAT2O1A5T2Q9FØO1S5T2R4A1U8R3AØWØD2W1T2O2A1U8R4A3U8R3A1WØD2W1T2O2A3U8U6Z5U7
53 DATAU8D4Q9A1O1Z5U7R4A5U8R3ZØT2U8BØU2W4F9R6B3X8ZØT2U8A8X2W4Y7O4A3U6ZØT2O2A5U8
54 DATAR3B6U8E1A4R2Y9T2U8B5X8A4U8U8C3H4U8G6Q9A6O1B6U8U8B3U6A4U8U8A8BØYØF4Q9A2O1
55 DATAB6U8Q2B8QØZ5N6U8Z3U2U8Y8R3D1U8U8A8R3DØU8YØB6U6S5T2R3B6U8X8Y9T2E1A4U8A3O1
56 DATAY9T2H6I1T2K6O4B1K6O4B5J6N3A2J6Q9A1U8Y9U2W4F8R6Y3X2J6X2W4XØO4X6U2J6A1A1Z5
```

```
Ø PRINT"(CLR)(RVS)(YEL)";:FORI=ØTO39:PRINT" ";:NEXT:V=49152:FORJ=1TO7:READA$:FOR
I=1TO36
1 POKEV,1Ø*(ASC(MID$(A$,I*2-1,1))-65)+VAL(MID$(A$,I*2,1)):V=V+1:NEXT:NEXT
2 V=832:FORI=VTO96Ø:POKEI,Ø:NEXT:FORI=ØTO2Ø:POKEV+I*3,255:NEXT:V=53248:POKE2,Ø
8 POKEV+16,2:POKEV+4,Ø:POKEV+21,7:Q=(PEEK(2)AND3):P(Q)=P(Q)+1:PRINT"(HOME)(RVS)"
P(2);
9 PRINTSPC(25);P(1):SYS49152:GOTO8
5Ø DATAQ9O3O1A1U8O1A3U8Q9AØO1D3U8O1C3U8O1C9U8Q9H2O1A2U8Q9D2O1AØU8Q2B3O2Y9A7O2Y8
51 DATAA7X2O2ZØA7X2O2N1A3O2N4A3Q2A4O2EØU8X2O2D9U8X2O2M8A3O2N7A3X2O2D2U8Q9R3O1A2
52 DATAT2O1A5T2Q9FØO1S5T2R4A1U8R3AØWØD2W1T2O2A1U8R4A3U8R3A1WØD2W1T2O2A3U8U6Z5U7
53 DATAU8D4Q9A1O1Z5U7R4A5U8R3ZØT2U8BØU2W4F9R6B3X8ZØT2U8A8X2W4Y7O4A3U6ZØT2O2A5U8
54 DATAR3B6U8E1A4R2Y9T2U8B5X8A4U8U8C3H4U8G6Q9A6O1B6U8U8B3U6A4U8U8A8BØYØF4Q9A2O1
55 DATAB6U8Q2B8QØZ5N6U8Z3U2U8Y8R3D1U8U8A8R3DØU8YØB6U6S5T2R3B6U8X8Y9T2E1A4U8A3O1
56 DATAY9T2H6I1T2K6O4B1K6O4B5J6N3A2J6Q9A1U8Y9U2W4F8R6Y3X2J6X2W4XØO4X6U2J6A1A1Z5
```

Es ist eine Nachahmung des ersten Telespiels überhaupt: eine einfache Ping-Pong-Simulation. Dabei übt gerade seine Einfachheit einen besonderen Reiz aus; vielleicht liegt dies auch in der »Überfrachtung« mancher professioneller Spiele mit Spezialeffekten begründet, worunter ja oftmals der eigentliche Spielwert leidet.

Gesteuert wird das Spiel mit zwei Joysticks (für jeden Spieler ein Joystick). Will man alleine spielen, kann man durch Einfügen neuer Zeilen eine Mauer aus reversen Leerzeichen bauen, von der der Ball abgelenkt wird. Man muß hierbei nur darauf achten, daß bis zur Spalte 29 der Ball nur nach rechts, ab der 31. Spalte der Ball nach links abgelenkt wird. Obwohl das Programm sehr kompakt und dadurch alles andere als transparent wirkt, lassen sich sogar bestimmte Parameter modifizieren:

- Die Geschwindigkeit des Balles wird durch POKE49335,V festgelegt, wobei 18 der Standardwert für V ist.
- Der Winkel des Balles wird durch POKE 49263,2 abgeflacht (normaler Wert ist 1).

Diese Änderungen sind am besten zwischen den Zeilen 2 und 8 einzubauen; vor dem Start durch RUN bleiben sie unberücksichtigt. Das Spiel kann durch <RUN/STOP> vorzeitig abgebrochen werden.

2.3 Funktionsdefinitionen

Das Basic 2.0 des C 64 unterstützt zwar weder Prozeduren noch Label, aber ein wichtiges Element der strukturierten Programmierung ist zumindest ansatzweise vorhanden: die benutzerdefinierte Funktion. Wenn von den allermeisten Anwendern kein Gebrauch der Anweisung DEF FN gemacht wird, so liegt dies vor allem daran, daß keine fertigen, jederzeit einzubauenden Funktionen bekannt sind. Dieser Abschnitt nennt einige Anwendungen von DEF FN, die Ihnen bestimmt das Programmierhandwerk erleichtern.

2.3.1 Der Fehler in der FRE-Funktion

Die FRE-Funktion, deren Parameter – Zahl oder String – frei wählbar ist und beim C64 keinen Einfluß auf das Funktionsergebnis hat, liefert oft merkwürdige Ergebnisse. Geben Sie nach Einschalten des C64, wo die Meldung »38911 BASIC BYTES FREE« erscheint, doch einmal PRINT FRE(0) ein. Sie erhalten eine negative Zahl! Diese Schwäche liegt in einem Programmierfehler der FRE-Routine (siehe »C64 für Insider«) begründet und wird durch folgende Funktionsdefinition ausgebügelt:

```
DEF FN FR(X) = FRE(0)-(65536*(FRE(0)<0))
```

Fortan liefert FN FR(0) immer das richtige Ergebnis. Auch hier ist das Argument der Funktion FR ohne Einfluß auf das Ergebnis anzugeben. Syntax-Vorschrift ist Vorschrift!

2.3.2 Codewandlung

Beim C64 sind die CHR$-Codes (ASCII-Codes) des Betriebssystems anders codiert als die Bildschirmcodes des Video-Chips, wie sie etwa im Bildschirmspeicher ab Adresse 1024 im Speicher stehen. Mit dem Problem, eine Konvertierung zwischen beiden Formaten durchzuführen, wurden schon viele Basic-Zeilen beschäftigt. Es geht jedoch auch durch Funktionsdefinitionen:

Wandlung von ASCII nach Bildschirmcode

```
DEF FN AB(A) = A + 33*(A=255) + 64*(A>63) + 32*(A<96) - 32*(A<160)
             + 64*(A>191)
```

Die Funktion FN AB(x) liefert zum ASCII-Code x den Bildschirmcode zurück. Infolgedessen ergibt FN AB(ASC("X")) den Bildschirmcode des Zeichens X.

Wandlung von Bildschirmcode nach ASCII

```
DEF FN BA(B) = B + 64 + 64*(B<64 AND B>31) + 32*(B<96 AND B>63)
```

FN BA(x) ist somit der ASCII-Code zum Bildschirmcode x, CHR$(FN BA(x)) das Zeichen, welches durch den Bildschirmcode x repräsentiert wird.
Diese beiden Funktionsdefinitionen berücksichtigen alle Sonderfälle, sind jedoch auf logische Fehler nicht vorbereitet:

- Die Parameter müssen im Bereich 0–255 liegen.
- Von Steuerzeichen oder undefinierten Codes wird kein korrekter Bildschirmcode errechnet.
- Bei reversen Bildschirmcodes (Codes > 127) ist vorher Bit 7 auszublenden:
  B=B AND 127:REM Reversflag aus Bildschirmcode B entfernen

- Fehlerhafte Angaben werden in der Regel nicht aussortiert, sondern bringen auch ein fehlerhaftes Ergebnis!

2.3.3 Zufallszahlen

Die meisten Zufallszahlen sollen im Bereich von 1 bis x liegen, also beispielsweise 1–100. Hierfür liefert die Funktion FN RD(x) eine Zufallszahl, die minimal 1 und maximal x beträgt:

```
DEF FN RD(X) = INT (RND(1)*X)+1
```

Somit würfelt PRINT FN RD(6) eine Zahl zwischen 1 und 6.

2.3.4 Ungerade oder gerade Zahl?

Die Funktion FN OD(x) ergibt 1 bei ungeraden, 0 bei geraden Zahlen:

```
DEF FN OD(X) = X AND 1
```

Um die gängigen Wahrheitswerte 0 und –1 zu erhalten, ist die Funktionsdefinition wie folgt zu modifizieren:

```
DEF FN OD(X) = -(X AND 1)
```

2.3.5 Uhrzeit

Nach

```
DEF FN DZ(HR) = INT((INT(HR)+(HR-INT(HR))/.6)*100+.5)/100
```

ergibt FN DZ(17.30) den Dezimalwert 17.5, der für Berechnungen besser geeignet ist.

Die umgekehrte Berechnung erledigt

```
DEF FN UR(DZ) = INT ((INT(DZ)+(DZ-INT(Dz))*.6)*100+.5)/100
```

PRINT FN UR(17.25) ergibt die normale Uhrzeit 17.15 Uhr, welche durch 17.25 dezimal dargestellt wird.

2.3.6 INT-Funktion berichtigt

Nicht nur die FRE-, sondern auch die INT-Funktion hat einen Programmierfehler. Für Mathematiker: Die INT-Funktion ist keineswegs die definitionsgemäße Integer-Funktion, sondern die Gaußklammerfunktion. Glücklicherweise ist INT im Handbuch richtig erklärt, nur stimmt

nicht, daß »negative Zahlen dem Betrag nach größer werden«, wenn man die Nachkommastellen abschneidet. Die im Basic-Interpreter definierte Funktion rundet nämlich alle Zahlen ab, anstatt die Nachkommastellen abzuschneiden. Aus INT(–1.23) wird somit –2, und nicht, wie es richtig wäre, –1! Es ist also Vorsicht geboten bei Programmen oder Rechnungen, die sich auf die Integerfunktion stützen und bei welchen ein negativer INT-Parameter vorliegt.
Bei kaufmännischen oder sonstigen Anwendungen des C64, von denen beileibe nicht nur das Erfolgserlebnis des Programmierers abhängt, wäre es nötig, die Funktion im Programm neu zu definieren:

```
DEF FN IN(X) = INT(ABS(X)) * SGN (X)
```

Dadurch wird der Absolutwert (Betrag) zwar mit der alten Integerfunktion behandelt; da INT bei positiven Zahlen – und ABS(x) ist immer positiv – die Nachkommastellen korrekt abschneidet, ging durch ABS(x) nur das Vorzeichen verloren. Durch die Nach-Multiplikation mit SGN(x) wird es wiederhergestellt.

2.3.7 Rundungsfehler behoben

Und wenn wir schon beim Ausmerzen aller Fehler des C64-Basic sind: Wie jedermann weiß, ist die Rechengenauigkeit des C64 teilweise unbefriedigend. Manchmal werden gerundete Ergebnisse angezeigt, die zwar richtig aussehen mögen, deren ganzzahliger Anteil, welcher durch INT ermittelt wird, allerdings um eins kleiner ist. Beispiel:

richtig: 1/50 * 100 = 2
falsch: INT(1/50 * 100) = 1 < > INT(2) !

Dies tritt durch die interne Speicherung von Rundungsbits auf; diese wiederum wird verhindert, indem man die Zahl erst mit STR$ in einen String umwandelt und anschließend durch VAL wieder eine numerische Zahl erzeugt. Wird dann die ganze Zahl ermittelt, entsteht das richtige Ergebnis

```
INT(VAL(STR$(1/50*100))) = 2
```

Definitionsgemäß müßten sich VAL und STR$ gegenseitig aufheben, wie man jedoch sieht, entsteht ein geringfügig verschiedenes Ergebnis, welches wir uns hier zunutzemachen.

2.3.8 Joystickabfrage

Jeder Neuling unter den C64-Fans wird sich früher oder später fragen, warum Commodore gleich zwei Joystick-Ports einbaute, es aber sträflich versäumte, das Basic des C64 um eine Funktion zur Abfrage dieser Joysticks zu bereichern. Auf dem C64 wurden solche Abfragen bisher mit langwierigen IF-THEN-Sequenzen über die PEEK-Funktion realisiert. Hier ist nun der viel elegantere Weg über eine benutzerdefinierte Funktion:

```
DEF FN JOY(X)=INT((LOG(255.5-(PEEK(56322-X)OR224)))/LOG(2)+2)
```

Anschließend liefert FN JOY(X) für Port x (1 oder 2) die Position des Joysticks:

1	Nullstellung
2	oben
3	unten
4	links
5	rechts
6	Feuerknopf

Über eine Anweisung wie

```
ON FN JOY(X) GOTO NULLSTELLUNG, OBEN, UNTEN, LINKS, RECHTS, FEUER
```

ließe sich dann äußerst schnell in die entsprechenden Unterprogramme verzweigen.

2.3.9 Exklusives Oder

Der Ausdruck »x OR y« ist erfüllt, wenn x, y oder beides »wahr« (–1) ist. Die EXOR-Verknüpfung ist nur dann gegeben, wenn entweder x oder y (aber nicht beides) zutrifft. Definiert man die Variablen X und Y, so ergibt sich für einen beliebigen Parameter E folgende EXOR-Definition in Basic:

```
DEF FN XO(E)=((NOT X) AND Y) OR ((NOT Y) AND X)
```

Beispiel:

`X=27:Y=15:PRINT FN XO(0)` ergibt: `20`

Im Gegensatz dazu liefert

`X=27:Y=15:PRINT X OR Y` das folgende Ergebnis: `31`

Belegt man X und Y mit Wahrheitswerten (0 oder –1, s. 2.6), so ist FN XO() auch darauf anwendbar.

2.4 Maschinenroutinen in Basic-Zeilen

Die Verbindung von Basic-Programmen mit Routinen in Maschinensprache ist eines der reizvollsten, aber sicher auch eines der schwierigsten Probleme, vor die uns der C64 stellt. Mit einem neuartigen Hilfsprogramm zeige ich Ihnen eine interessante Alternative zum Nachladen von Maschinenroutinen.

2.4.1 Das Konzept

Üblicherweise werden Maschinenprogramme zum Aufruf vom Basic-Programm nachträglich eingelesen. Dieses »Nachladen« von Maschinenroutinen hat jedoch gravierende Nachteile:

- Der Start der Routinen liegt fast immer bei 828 ($033c) oder 49152 ($c000). Will man mehrere Routinen gleichzeitig verwenden, müssen sie oftmals auf andere Adreßbereiche umgeschrieben werden.
- Bei einem Reset verschwinden Programme aus dem Kassettenpuffer ($033c–$03fb). Sie müssen nachgeladen werden, selbst wenn das Basic-Programm durch einen OLD-Befehl, wie manche Basic-Erweiterungen ihn besitzen, gerettet werden kann.
- Das Nachladen der Routinen ist eine Prozedur, die zusätzlich Zeit in Anspruch nimmt und das Programmieren einengt.
- Wird eine Routine vergessen oder durch Pokes gestört, kann es zu Abstürzen kommen, die unter Umständen stundenlange Arbeit unwiederbringlich vernichten.

Diesen Ärger gibt es nicht mehr, wenn die Unterroutinen, die für ein Programm benötigt werden, am Anfang des Basic-Programms stehen und infolgedessen »gleichzeitig« mit ihm geladen werden. Mit »SYS Variable« und gegebenenfalls weiteren Parametern erreichen Sie sie jederzeit. Das Programm »MPRG IM BASIC« übernimmt diese Einbindung der Maschinenroutinen in Basic-Zeilen. Dabei sind einige Vorgaben zu beachten:
Der Basic-Interpreter verarbeitet beim Programmieren, Laden und Listen jeweils Zeilen mit bis zu 255 Zeichen. Darin enthalten sind zwei Byte als Zeiger auf den Beginn der nächsten Zeile, zwei Byte für die Zeilennummer und das Schlußbyte ($00). Beim Ablauf eines Basic-Programms wird bei REM die nächste Zeile angesprungen. Folglich können hinter einem REM, das eine Zeile einleitet, 249 beliebige Byte stehen, die nicht vom Basic-Interpreter ausgeführt werden und somit nicht der üblichen Basic-Syntax des C 64 entsprechen müssen. Anstelle der üblichen Bemerkungen im Klartext können diese Bytes folglich auch zu einem Maschinenprogramm gehören. Dieser Trick wurde bei einigen der Einzeiler aus 2.1 eingesetzt.
Mit solchen mehr oder weniger langen REM-Zeilen, die Maschinenprogramme anstelle von Kommentaren enthalten, wäre also der Platz für Routinen im Basic-Programm geschaffen. Für diese Routinen müssen aber noch zwei Regeln beachtet werden:

1. Es darf kein Null-Byte vorkommen, da eine Null das Zeilenende markiert und bei Einbindung oder Löschen anderer Zeilen die REM-Zeile hier abgeschnitten wird. Durch INC, DEC (Selbstmodifikation des Programms, siehe Kapitel 10) oder Laden aus der Zeropage läßt sich das Auftreten von Nullbytes jedoch mit vertretbarem Aufwand umgehen; so könnte sich ein Programm zu seinem Beginn durch INC entsprechend »bearbeiten«, daß alle in ihm stehenden Nullbytes – welche zuvor als $ff vermerkt sind – durch INC hergestellt werden. Gleiches gilt für DEC bei $01 als $00-Ersatz.
2. Wer die REM-Zeilen wahlweise einsetzen will, darf – abgesehen vom Aufruf der ROM-Routinen – nur bedingte Verzweigungen (Branches) verwenden.

Eine direkte Änderung dieser REM-Zeilen auf dem Bildschirm ist nicht allein wegen einer möglichen Überlänge, sondern auch wegen der eigenartigen Ausgabe unmöglich. Wenn nämlich der Interpreter beim Listen ein Byte keinem Zeichen zuordnen kann, behandelt er es als Token und druckt das Basic-Befehlswort aus. Zum Teil führt er sogar Steuerzeichen aus (zum Beispiel setzt der Wert 31 die Schriftfarbe auf Blau). Verwirrende optische Effekte, die teilweise wie Systemabstürze wirken, sind die Folge. <RUN/STOP RESTORE> bringt alles wieder in einen geordneten Zustand.
Wer hingegen einen Listschutz vortäuschen will, kann das Resultat dieser Pseudo-Abstürze so stehenlassen und ab der ersten Basic-Programmzeile listen. Eleganter ist aber, eine »KORREKTURZEILE« (s.u.) anzuhängen, die den Bildschirm »aufräumt«. Beim Wert 204 ($CC) jedoch stoppt das Listen unweigerlich und es erscheint die Meldung SYNTAX ERROR.

2.4.2 Die Programme

Das Programm »MPRG IM BASIC« übernimmt die Einbindung einer Routine. Dieses Basic-Programm verändert sich während des Ablaufs. Daher darf es ohne genaue Kenntnis überhaupt nicht geändert werden. Jede unbedachte Änderung führt zur Fehlfunktion! Der Originalzustand besteht nur vor dem ersten Start. Dies gilt auch für die weiteren Hilfsprogramme:

»MERGE« ist eine Routine zum Anhängen eines Basic-Programms an das im Speicher befindliche Programm. Diese Routine eignet sich hervorragend zu einem ersten Test von »MPRG IM BASIC« und wird des weiteren für alle folgenden Operationen als Hilfsroutine benötigt.

»STARTADRESSEN« ist ein Basic-Programm zur Ermittlung der Einsprungadressen mehrerer Routinen in REM-Zeilen. Es beginnt mit der Zeile 60000, numeriert die Zeilen fortlaufend ab 1, gibt die Adressen dezimal auf den Bildschirm und Drucker (Geräteadresse 4) aus und vermag sich nach geleisteter Arbeit selbst zu löschen.

»KORREKTURZEILE« ein Einzeiler, enthält hinter einem REM die Bytes, die nach dem Listen der REM-Zeilen den Normalzustand des Bildschirms wiederherstellen.

2.4.3 Anwendung der Programme

Nach dem Start gibt »MPRG IM BASIC« eine Kurzinformation aus und möchte wissen, ob sich das umzusetzende Programm, dessen Startadresse beliebig ist, auf der Diskette befindet. Es wird dann nach dem Programmnamen gefragt. Sie können diesen mit den von Floppy-Befehlen zugelassenen Jokern und »wild cards« (»*« und »?«) abkürzen. Nach der Eingabe wird das Programmfile geöffnet. Der Status der Diskette wird angezeigt. Bei Fehlermeldungen ist die Leertaste zu drücken.

Bei der darauffolgenden Eingabe können Sie eine Zeilennummer von 1 bis 99 wählen. Jetzt wird das Programm geladen und die Länge der Routine überprüft. Es erscheint ein Vorschlag für den Namen der REM-Zeile, der sich aus Zeilennummer und eingegebenem Namen zusammensetzt. Diesen Vorschlag können Sie unverändert übernehmen oder eigene Eingaben tätigen. Vor dem Speichern ist Gelegenheit zum Diskettenwechsel. Es wird auch hier der Diskettenstatus angezeigt, so daß ein irrtümliches Überschreiben verhindert ist. Das Speichern ist mehrmals möglich.
Nun können Sie die nächste Routine bearbeiten lassen oder erst das Umwandlungsergebnis ansehen. Entschließen Sie sich zu letzterem, wird das Programm gelöscht und die neue REM-Zeile geladen. Die Startadresse liegt bei 2054.

Zum Einbinden einer REM-Zeile mit Maschinenprogramm in Ihr Basic-Programm gehen Sie wie folgt vor:

1. Die REM-Zeile mit der MERGE-Routine laden und mit SYS 2054 starten.
2. Nacheinander die gewünschten Routinen mit Hilfe von MERGE laden, wobei die Reihenfolge hier beliebig ist.
3. Nach den REM-Zeilen die »KORREKTURZEILE« mit MERGE anhängen und die Anzahl aller Zeilen merken.
4. Das Hauptprogramm mit MERGE nachladen. Die erste Zeilennummer muß größer als die Anzahl der zuvor geladenen Zeilen sein. Die höchste Zeilennummer hat kleiner als 60000 zu sein.
5. Die Hilfsroutine »STARTADRESSEN« anhängen und mit RUN 60000 starten. Nach dem Ausdruck der ermittelten Einsprungadressen können Sie dieses Programm löschen. Prüfen Sie jedoch zuvor, ob Sie die MERGE-Routine noch benötigen. Sie steht jetzt in Zeile 1. Sobald Sie eine REM-Zeile gelöscht haben, müssen Sie »STARTADRESSEN« erneut starten.
6. Den REM-Zeilen sollte eine Zeile folgen, in der Basic-Variablen als Einsprungadressen für die Routinen definiert werden, zum Beispiel ME=2054 für MERGE. SYS ME ruft dann die MERGE-Routine auf.

2.4.4 Beispiele und Vorschläge

Auf der Programmdiskette befindet sich auch eine Auswahl von weiteren Routinen in Maschinensprache. Zusätzlich wurden sie in Basic-Zeilen umgesetzt und sie sind außerdem in einem eigenen Demonstrationsprogramm namens »DEMO ZU MPRG« enthalten. Das File »Z 1000« wird von »DEMO ZU MPRG« herangezogen. Am besten geben Sie als erstes

```
LOAD "DEMO ZU MPRG",8
```

und RUN ein.

Aus den Beispielen, vor allem "DEMO ZU MPRG", können Sie bestimmt lernen. Dabei ist noch folgendes zu beachten:
Es lassen sich auch Maschinenprogramme mit einer Länge von über 249 Byte in Basic-Programme einbinden. Das geht aber erst bei fertigen Programmen, die dann ohne Zerstörung des Maschinenprogramms nicht änderbar sind. Im Prinzip ist das Verfahren ähnlich, aber schwieriger anzuwenden.

2.5 PEEK-, POKE- und SYS-Befehle zum C64

Die Befehle PEEK, POKE und SYS arbeiten sehr maschinennah, d.h., sie kommunizieren direkt mit dem Betriebssystem und dem Basic-Interpreter. Dadurch lassen sich viele interessante Effekte hervorrufen, und es gibt mittlerweile ellenlange PEEK- und POKE-Listen. Im gesamten Verlauf des Buches haben und werden Sie Dutzende von PEEK-, POKE- und SYS-Anweisungen erfahren; dieser Abschnitt setzt sich jedoch noch ein weiteres Ziel, welches in bisheriger Literatur noch nicht verfolgt wurde: Hier stehen die wichtigsten, nützlichsten und einfachsten PEEKs und POKEs; eben genau das, was man beim Programmieren tatsächlich braucht!

▶ `POKE 0,0`

Mit diesem Trick können Sie jeden verblüffen, der die Bedeutung der Adresse 1 des C64 kennt. Nach POKE 1,0 stürzt der C64 normalerweise ab. Folgender POKE trifft Vorsorge: POKE 0,0
Denn die Adresse 0 ist das Datenrichtungsregister für Adresse 1; wenn Sie dieses auf 0, also auf Eingang, schalten, werden sämtliche POKE-Zugriffe auf Adresse 1 ignoriert. Durch Drücken von <RUN/STOP RESTORE> wird das Datenrichtungsregister wieder auf den Normalwert 47 geschaltet.

▶ `PRINT PEEK(10)`

Ausgabe 0:	letzte Ladeoperation LOAD
Ausgabe 1:	letzte Ladeoperation VERIFY

▶ `PRINT PEEK(17)`

0:	letzte Variablenzuweisung über INPUT
64:	letzte Variablenzuweisung über GET
152:	letzte Variablenzuweisung über READ

▶ `POKE 19,64`

Durch POKE 19,64 wird beim nächsten INPUT kein Fragezeichen mehr ausgegeben. Allerdings kann man durch Drücken von <RETURN> nicht mehr in die nächste Zeile gelangen. Es empfiehlt sich daher, nach dem INPUT-Befehl dies wieder mit POKE 19,0 rückgängig zu machen.

▶ `POKE 22,35`

Zeilennummern bei LIST erscheinen nicht mehr.

▶ `POKE 24,0`

stellt nach einem FORMULA TOO COMPLEX ERROR den Normalzustand her.

▶ `PRINT PEEK(43)+256*PEEK(44)`

Ausgabe der Adresse, von wo aus das Basic-Programm gespeichert ist.
Nach POKE 44,9:LOAD "NAME",8,1...
...verliert fast jedes mit Autostart versehene Basic-Programm seine Wirkung. Sobald Sie sich dann im Eingabemodus wiederfinden, können Sie nach POKE 44,8 das Programm abspeichern.

▶ `PRINT PEEK(45)+256*PEEK(46)`

Ausgabe der Adresse, bis wohin das Basic-Programm reicht.

▶ `PRINT PEEK(47)+256*PEEK(48)`

Ausgabe der Adresse, bis wohin die normalen Variablen reichen.

▶ `PRINT PEEK(49)+256*PEEK(50)`

Ausgabe der Adresse, bis wohin die Felder reichen.

▶ `PRINT PEEK(55)+256*PEEK(56)`

Ausgabe der Adresse, bis wohin der Basic-Arbeitsspeicher reicht.

▶ `PRINT PEEK(57)+256*PEEK(58)`

Ausgabe der Nummer der momentan bearbeiteten Programmzeile.

▶ `PRINT PEEK(59)+256 *PEEK(60)`

Ausgabe der Nummer der letzten bearbeiteten Programmzeile.

▶ `PRINT PEEK(61)+256*PEEK(62)`

Ausgabe der Nummer der CONT-Fortsetzungszeile.

▶ `PRINT CHR$(PEEK(69)AND127) + CHR$(PEEK(70)AND127)`

Ausgabe des zuletzt benutzten Variablennamen.

▶ `POKE 120,2`

Danach nimmt der C 64 keine Basic-Befehle mehr an.

▶ `WAIT 145,16,16`

Warten auf Feuerknopf von Joystick in Port 1.

▶ `POKE 157,128`

Vortäuschen des Direktmodus innerhalb von Programmen, um Meldungen wie SEARCHING, LOADING usw. auch in Programmen ausgeben zu lassen.

▶ `POKE 157,0`

Vortäuschen des Programm-Modus im Direktmodus, entgegengesetzte Wirkung vom letzten POKE.

▶ `PRINT PEEK(182)`

Ausgabe der Anzahl der Zeichenlesefehler bei RS 232.

▶ `PRINT PEEK(183)`

Ausgabe der Länge des letzten Filenamens.

▶ `PRINT PEEK(184)`

Ausgabe der laufenden Filenummer.

▶ `PRINT PEEK(185)`

Ausgabe der laufenden Sekundäradresse.

▶ `PRINT PEEK(186)`

Ausgabe der laufenden Geräteadresse.

▶ `PRINT PEEK(187)+256*PEEK(188)`

Ausgabe der Adresse, von wo ab sich der Filename befindet.

▶ `SYS 63123`

Ausgabe des letzten Filenamens.

▶ `POKE 198,0`

Löschen des Tastaturpuffers.

▶ `POKE 199,1`

Reverse Zeichenausgabe.

▶ `POKE 199,0`

Normale Zeichenausgabe (nicht revers).

▶ `PEEK (203)`

Zwischencode der aktuell gedrückten Taste (64=keine Taste).

▶ `POKE 204,0`

schaltet den Cursor jederzeit (auch bei GET) an.

▶ `POKE 207,0:POKE 204,1`

Rückgängigmachung von POKE 204,0.

▶ `PRINT PEEK(214)`

Ausgabe der Nummer der Zeile, in welcher sich der Cursor befindet; dabei werden die Zeilen von 0 bis 24 (=25 Zeilen) durchgezählt.

▶ `POKE 214,Z:POKE 211,S:SYS 58640`

Diese Befehlssequenz setzt den Cursor unmittelbar auf die von Z uns S angegebene Zeilen- und Spaltenposition (Z von 0 bis 24, S von 0 bis 39).

▶ `PRINT PEEK(215)`

Ausgabe des zuletzt eingegebenen Zeichencodes.

▶ `PRINT PEEK(641)+256*PEEK(642)`

Ausgabe der Adresse, von wo ab der für Basic nutzbare RAM-Bereich beginnt.

▶ `PRINT PEEK(643)+256*PEEK(644)`

Ausgabe der Adresse, bis wohin der für Basic nutzbare RAM-Bereich steht.

▶ `POKE 641,AL:POKE 642,AH:SYS 58260`

Die Anfangsadresse für Basic-Programme wird auf den Wert AL+256*AH festgelegt.

▶ `POKE 643,EL:POKE 644,EH:SYS 58260`

Die Endadresse für den Basic-Arbeitsspeicher wird auf den Wert EL+256*EH festgelegt.

▶ `PRINT PEEK(646)`

Anzeige der Nummer (0–15) der momentanen Zeichenfarbe.

▶ `POKE 646,F`

Einstellen der Zeichenfarbe F, wobei F der Farbnummer (0–15) entspricht.

▶ `POKE 649,0`

Abstellen der Tastatur (Vorsicht!).

▶ `POKE 650,128`

Tasten-Wiederholungs-Automatik (Repeat) für alle Tasten.

▶ `POKE 650,64`

Tasten-Wiederholungs-Automatik (Repeat) für keine Taste.

▶ `POKE 650,0`

Tasten-Wiederholungs-Automatik (Repeat) nur für <INST/DEL>, <Leertaste> und <CRSR>.

▶ `POKE 651,x`

Setzen des Zählers für die Repeat-Geschwindigkeit (s. POKE 650,...).

▶ `POKE 652,x`

Setzen des Zählers für die Verzögerung, bis ein Repeat (s. POKE 650,...) eintritt.

▶ `PRINT PEEK (653) AND 7`

Anzeige von 0:	keine der Tasten <SHIFT>, <C=>, <CTRL> gedrückt
Anzeige von 1:	<SHIFT> gedrückt
Anzeige von 2:	<C=> gedrückt
Anzeige von 3:	<SHIFT + C=> gedrückt
Anzeige von 4:	<CTRL> gedrückt
Anzeige von 5:	<CTRL + SHIFT>
Anzeige von 6:	<CTRL + C=>
Anzeige von 7:	<CTRL + SHIFT + C=>

▶ `POKE 657,128`

Umschaltungsverriegelung für <SHIFT C=>.

▶ `POKE 657,0`

Aufheben der Umschaltungsverriegelung (siehe vorheriger Befehl).

▶ `POKE 768,61`

Innerhalb eines Programms wird dadurch jede Fehlermeldung unterdrückt. Beispiel:

```
10 POKE 768,61:FOR F=-10 TO 10:PRINT 1/F:NEXT:POKE 768,139
```

Die Fehlermeldung DIVISION BY ZERO ERROR würde bei F=0 erscheinen. Durch den vorherigen POKE wird sie – nur in Programmen! – unterdrückt.

▶ `POKE 768,139`

läßt Fehlermeldungen wieder zu (s. letzter POKE).

▶ `POKE 768,226:POKE 769,252`

Programmende führt zu Reset.

▶ `POKE 775,200`

LIST-Schutz einschalten (LIST funktioniert nicht).

▶ `POKE 775,167`

LIST-Schutz ausschalten.

▶ `POKE 774,226:POKE 775,252`

LIST führt zu Reset.

▶ `POKE 788,52`

<RUN/STOP> ausschalten.

▶ `POKE 788,49`

<RUN/STOP> reaktivieren.

▶ `POKE 792,193`

<RUN/STOP RESTORE> disaktivieren, jedoch nicht <RUN/STOP> alleine (siehe vorheriger POKE).

▶ `POKE 792,226:POKE 793,252`

RESTORE-Taste löst fortan Reset aus (nur kurz drücken!).

▶ `POKE 801,0:POKE 802,0`

Das Speichern eines Programms wird unmöglich.

▶ `POKE 808,225`

<RUN/STOP RESTORE> wirkungslos, LIST arbeitet nicht korrekt.

▶ `POKE 808,237`

<RUN/STOP RESTORE> reaktiviert.

▶ `POKE 53265,11`

Bildschirminhalt unsichtbar machen, ohne daß dessen Inhalt verlorengeht.

▶ `POKE 53265,27`

Bildschirminhalt wieder sichtbar machen.

▶ `WAIT 56320,16,16`

Warten, bis beim Joystick in Port 2
– *der Feuerknopf gedrückt wird,*

▶ `WAIT 56320,4,4`
– *Linksbewegung vorgenommen wird,*

▶ `WAIT 56320,1,1`
– *Bewegung nach oben erfolgt,*

▶ `WAIT 56320,2,2`
– *Bewegung nach unten erfolgt,*

▶ `WAIT 56320,8,8`
– *Rechtsbewegung vorgenommen wird,*

▶ `WAIT 56321,x,x`
– *Abfragen analog zu 56320, aber diesmal für Joystick in Port 1.*

▶ `SYS 64738`

Reset,

▶ `SYS 64767`

Verkürzter (schnellerer) Reset, bei welchem die Bildschirmfarben erhalten bleiben, aber viele Basic-Erweiterungen (Simon's Basic etc.) abgeschaltet werden (was oft durch SYS64738 nicht möglich ist).

2.6 Mathematische Auswertung logischer Ausdrücke

Eine sehr nützliche Eigenschaft des Basic 2.0, welche auch in anderen Basic-Versionen vorhanden ist, wird im C64-Handbuch verschwiegen und in bisheriger Literatur, von Ausnahmen abgesehen, nur unzureichend erklärt: die numerische Auswertung von Bedingungen.
Damit ist gemeint, daß man IF-ähnliche Abfragen in mathematische Formeln einbinden kann, um IF-THEN-Konstruktionen zu umgehen.

2.6.1 Experimente

Zur Demonstration geben Sie bitte folgende Befehle im Direktmodus ein, auch wenn Sie noch befürchten, daß dabei ein SYNTAX ERROR entsteht:

PRINT 1<2	Antwort des C64: –1
PRINT 1=2	Antwort des C64: 0
PRINT 1>2	Antwort des C64: 0
PRINT 1< >2	Antwort des C64: –1

Bei genauerer Betrachtung erkennt man, daß der Computer dann "–1" ausgibt, wenn der hinter PRINT stehende Ausdruck wahr ist (wie "1<2"). Die Antwort ist 0, wenn der Ausdruck falsch ist (wie "1=2").

Offensichtlich wertet der Computer diese mathematischen Aussagen aus, denn das Basic 2.0 errechnet für einen logischen Ausdruck, erkennbar an einem Gleich- oder Ungleichheitszeichen, einen Zahlenwert, welcher den Wahrheitswert (–1 für »wahr« oder 0 für »falsch«) wiedergibt.
Setzt man einen logischen Ausdruck in Klammern, läßt sich sogar mit der mathematischen Bewertung von 0 oder –1 weiterrechnen:

```
PRINT 5*(1<2)
```

ergibt »–5«, da »(1 < 2)« wahr ist und somit den Wert »–1« hat, welcher mit 5 multipliziert »–5« ergibt. Im Gegensatz dazu liefert

```
PRINT 5*(1>2)
```

den Wert »0«, da »(1 > 2)« falsch ist und somit den Wert »0« hat, welcher mit 5 multipliziert »0« bleibt.
In diesem Abschnitt erfahren Sie nun, wie man diese Eigenheit des Basic 2.0 gewinnbringend einsetzt. In zahlreichen Einzeilern ist Ihnen diese Technik jetzt schon über den Weg gelaufen, ich schulde Ihnen also gewissermaßen eine Erläuterung.
Dabei kommt es für Sie nicht so sehr darauf an, unbedingt zu verstehen, warum dieser oder jener Trick funktioniert (dazu muß man einiges an mathematischem Verständnis mitbringen), sondern vielmehr darauf, daß Sie einen Eindruck von der Praxisanwendung bekommen. Sollten Sie also die mathematische Beweisführung nicht völlig verstehen, können Sie diese ohne Bedenken überfliegen.

2.6.2 IF-Abfragen vorbereiten – IF für Insider

Sind Sie schon bei der Analyse eines Basic-Programms über eine Anweisung wie

```
IF A THEN ...
```

gestolpert und haben sich gewundert, warum der IF-Befehl eine Variable (A) anstelle eines logischen Ausdrucks (wie »A > 5«) abfragt? Dann erfahren Sie jetzt, warum diese auf den ersten Blick unsinnige Anweisung durchaus ihre Berechtigung hat.
Betrachtet man die Funktionsweise des IF-Befehls (siehe »C 64 für Insider«), arbeitet er nach folgendem etwas vereinfachten Schema:

Schritt 1: Zahlenwert (!) holen

Wie wir soeben gesehen haben, werden Vergleichsoperationen mit »=«, »<«, »>« und deren Kombinationen automatisch ins Zahlenformat umgewandelt. Deshalb muß der IF-Befehl nicht selbst prüfen, ob die ihm folgende Bedingung wahr ist, – wie man es vermuten würde –, sondern nur die universelle Routine aufrufen, die einen Zahlen-Ausdruck auswertet. Kurz: Der Ausdruck zwischen IF und THEN wird zunächst nicht anders behandelt als eine Zahlenangabe nach einer Variablenzuweisung oder einem PRINT-Ausdruck. Wenn man diesen wichtigen Gedankenschritt nachvollzieht, hat man das Prinzip unseres weiteren Vorgehens schon begriffen.

Schritt 2: Zahlenwert weiterverarbeiten

Ist der Wert 0, so wird die Bearbeitung nicht nach THEN fortgesetzt, sondern in der nächsten Basic-Programmzeile. Ist der Wert nicht 0 (z.B. »–1«), werden alle Befehle hinter THEN ausgeführt, indem der erste Befehl hinter THEN ähnlich GOTO angesprungen wird.
Jetzt ist Ihnen sicher klar, weshalb folgender IF-Befehl immer den Text »JA« ausgibt:

```
IF -1 THEN PRINT "JA"
```

Dies gilt auch für

```
A=-1:IF A THEN PRINT "JA"
```

Entsprechend würde folgender IF-Befehl nie in den THEN-Teil springen und dadurch keinen Text ausgeben, weil 0 für eine unerfüllte IF-Bedingung steht:

```
IF 0 THEN PRINT "JA"
```

oder demzufolge

```
A=0:IF A THEN PRINT "JA"
```

An diesen unsinnigen Beispielen kann man sehen, daß es möglich ist, den Wahrheitswert einer Aussage zuerst zu bestimmen, in einer Variablen zu speichern und erst im Bedarfsfall mit IF weiterzuverarbeiten.
Etwas umfangreicher ist hierzu folgendes Beispielprogramm, das ich Sie einzugeben bitte (da Sie aus dem Abtippen lernen werden, befindet sich das Programm nicht auf der Masterdiskette):

```
10 INPUT "BITTE ZWEI ZAHLEN:";A,B
20 V=(A=B)
30 PRINT "VERGLEICHSERGEBNIS:";V
40 PRINT "DIE ZWEI ZAHLEN SIND:";
50 IF V     THEN PRINT "GLEICH"
60 IF NOT V THEN PRINT "VERSCHIEDEN"
```

Dieses Programm fordert Sie zur Eingabe von zwei Zahlen auf und sagt Ihnen dann, ob diese gleich oder verschieden sind. Dabei wird nicht in einer IF-Abfrage geprüft, ob A und B gleich sind, sondern unmittelbar nach der Eingabe der Vergleich durchgeführt (Zeile 20). Das Ergebnis (gleich/verschieden) kommt in die Variable V, die als »Vergleichsergebnis« ausgegeben wird (Zeile 30), um das Verständnis zu erleichtern. Die beiden IF-Befehle in den Zeilen 50/60 prüfen dann nur noch den Wert der Variablen V.

Der große Vorteil liegt bei dieser Methode darin, daß die Variable V das Vergleichsergebnis über eine beliebig lange Zeit erhält und eine IF-Verarbeitung auch später erfolgen kann. Nehmen wir an, die Variablen A und B ändern sich, aber im Programm soll später ein Vergleich der alten Werte durchgeführt werden; ohne V stünden wir diesem Problem recht hilflos gegenüber.

Folgendes Programm zeigt dies; es dient dazu, sich selbst bei Eingabe von »J« zu listen:

```
10 INPUT "LISTEN (J/N)";A$
20 V=(A$="J"):REM A$ mit "J" vergleichen, Ergebnis in Variable V
30 A$="X":REM A$ neubelegen
40 IF V THEN LIST:REM V abfragen
```

Der Fachausdruck für die Variable V in diesem Programm lautet »Flag« (Flagge). Damit werden alle Variablen oder Speicherplätze bezeichnet, die in einem Programm keine Daten im herkömmlichen Sinn (Texte, Zahlenmaterial) enthalten, sondern ausschließlich als Informationsquelle dienen, ob eine bestimmte Programmfunktion (im Beispiel: LIST) auszuführen ist oder nicht. Auch das Betriebssystem und der Basic-Interpreter haben Flags; so sagt beispielsweise Adresse 650 aus, ob ein bestimmter Tastatur-Repeat besteht oder nicht (siehe 2.5).

Die Bedeutung des Begriffs »Flag« sollten Sie sich auch im Hinblick darauf merken, daß er in einigen Programmdokumentationen, welche noch folgen werden, Verwendung finden wird.

2.6.3 Addition/Subtraktion mit Ober- und Untergrenze

Um eine Variable X um 10 zu erhöhen, schreibt man einfach »X=X+10«. Soll dabei X niemals den Wert 100 erreichen oder überschreiten, so hat X vor der Addition logischerweise kleiner als 90 zu sein. Mit Hilfe von IF...THEN ist eine leichte und übersichtliche Formulierung möglich:

```
IF X<90 THEN X=X+10
```

Soll 100 hingegen die maximale Obergrenze darstellen, ist auch bei X=90 die Addition von 10 zulässig:

```
IF X<=90 THEN X=X+10
```

Soweit nichts Neues. Es besteht aber die Möglichkeit, unter Anwendung unseres bislang angeeigneten Wissens dasselbe Ziel ohne IF...THEN, also mit einem einzigen Befehl, zu erreichen:

```
X=X-10*(X<90)
```

beziehungsweise

```
X=X-10*(X<=90)
```

Dies ist zunächst etwas unverständlich; vor allem könnte der Operator »–« stören, wenn man bedenkt, daß im Endeffekt eine Addition durchgeführt werden soll. Ein mathematisch angelegter Beweis für die Richtigkeit dieses Ausdrucks ist erforderlich und dem Verständnis dienlich. Stören Sie sich bitte nicht an dem wissenschaftlichen Touch.

Beweis: "`X=X-10*(X<90)`" entspricht "`IF X<90 THEN X=X+10`"

Während »X=X+10«, der THEN-Teil des IF-Ausdruckes, immer gleich arbeitet, müssen wir bei »X=X–10*(X<90)« zwei unterschiedliche Behandlungen betrachten: Entweder ist »X<90« wahr, zählt also in der Rechnung als »–1«, oder falsch, zählt also als »0«.

Fall 1: »X<90« ist wahr. Bei der IF-THEN-Lösung wird 10 addiert. Trifft dies auch auf »X=X–10*(X<90)« zu?
Wir können für diesen Spezialfall den Klammerausdruck durch »–1« ersetzen, wie es der Basic-Interpreter schließlich ebenfalls tut:

```
X=X-10*(-1)
```

Das »10*(–1)« können wir ausmultiplizieren in »(–10)«

```
X=X-(--10)
```

Bei Auflösung der Klammern ergibt »– –« den Operator »+«:

```
X=X+10
```

Folgerung: Ist »X<90«, so wirkt »X=X–10*(X<90)« genauso wie »IF <90 THEN X=X+10«!
Nun bleibt noch nachzuweisen, daß dies auch für die andere Situation zutrifft.

Fall 2: »X<90« ist falsch. Bei der IF-Lösung wird nichts unternommen, X ändert sich dann also nicht.
In diesem Fall wird der Klammerausdruck »(X<90)« als »0« behandelt:

```
X=X-10*(0)
```

Rechnet man weiter, ergibt sich

```
X=X-0
```

oder:

```
X=X (X wird nicht verändert)
```

Die Subtraktion von 0 hat keinen Einfluß auf das Ergebnis, demnach wird X – wie bei der IF-THEN-Lösung, nicht beeinflußt. Folglich arbeitet »X=X–10*(X<90)« in jeder Situation wie »IF X<90 THEN X=X+10«.
Folgendes Demoprogramm soll dies veranschaulichen, indem es von 0 an wiederholt diesen Ausdruck durchführt:

```
10 X=0
20 X=X-10*(X<90):REM entspricht: IF X<90 THEN X=X+10
30 PRINT X
40 FOR I=1 TO 250:NEXT:REM kleine Verzögerung
50 GOTO 20
```

Wie man sieht, läßt sich eine IF-THEN-Anweisung auf diese trickreiche Weise einsparen. Es soll aber nicht verschwiegen werden, daß dies durch eine geringere Übersichtlichkeit erkauft wird.
Es sei noch kurz eine entsprechende Anwendung für die Subtraktion gezeigt. Diesmal soll eine Zahl von 100 in 10er-Schritten verringert werden, darf aber vor Subtraktion niemals kleiner als 10 sein, um ein positives Subtraktionsergebnis zu gewährleisten:

```
10 X=100
20 X=X+10*(X>10):REM entspricht: IF X>10 THEN X=X-10
30 PRINT X
40 FOR I=1 TO 250:NEXT
50 GOTO 20
```

In Zeile 20 haben sich jetzt Operator (»–« wurde zu »+«) und die Bedingung (»X>10« statt »X<90«) geändert; beide Lösungen basieren aber auf demselben Prinzip.
Zugegeben, diese Programmiertechnik ist sehr kompliziert und kann leider nur auf dem mathematischen Weg erläutert werden. Aber es spielt, wie schon gesagt, keine Rolle, ob Sie den mathematischen Beweis verstanden haben. Wichtig ist nur, daß Sie einen Einblick in diese trickreiche Programmierung bekommen haben. In späteren Kapiteln wird dies noch einmal aufgegriffen und an bestimmten Spezialfällen so ausführlich zerpflückt, daß bestimmt keine Verständnisschwierigkeiten auftreten. Wenn Sie aber sehen, wie mächtig manche Einzeiler nur aufgrund dieser Programmiertechnik sind, wissen Sie sicher den großen praktischen Nutzen einzuordnen. Und letztlich zählt das Ergebnis!

2.6.4 Eine interessante Anwendung

Soll eine Variable ihren Wert in Abhängigkeit von einer anderen erhalten, ohne daß eine mathematisch formulierbare Proportion besteht, benötigt man für jede Möglichkeit eine Zeile:

```
10 IF A= 5 THEN B=3261
20 IF A= 8 THEN B=7901
30 IF A= 9 THEN B=2079
40 IF A=17 THEN B= 681
50 IF A=99 THEN B=  -3
```

Durch clevere Anwendung der Errungenschaften dieses Kapitels kann man das Programm auf eine einzige Zeile verkürzen:

```
10 B=0-3261*(A=5)-7901*(A=8)-2079*(A=9)-681*(A=17)+3*(A=99)
```

Diese Methode spart Zeit, Speicherplatz und Tipparbeit; sie erhöht darüber hinaus die Übersichtlichkeit eines Listings, da der Lesende durch die fünf Zuweisungszeilen nicht aus dem Zusammenhang gerissen wird.
Oft will man auch den Wert einer Variablen abhängig von ihrem alten Wert ändern. Zum Beispiel: Wenn X=8, soll nun X=5 werden und umgekehrt. Normalerweise benötigt der Programmierer zwei Zeilen:

```
10 IF X=8 THEN X=5:GOTO 30
20 IF X=5 THEN X=8
```

Wendet man hier die ABS-Funktion mit Konsequenz an, reicht wiederum eine Zeile aus:

```
10 IF X=5 OR X=8 THEN X=ABS(X-8)+5
```

Zur Erklärung: ABS(x) entfernt das Vorzeichen von X, liefert also immer einen positiven Wert. Rechnen wir die Beispiele X=5 und X=8 durch:

```
X=5 -> X=ABS(5-8)+5
=ABS(-3)  +5
=        3+5
=          8
X=8 -> X=ABS(8-8)+5
=ABS(0)   +5
=        0+5
=          5
```

3 Basic-Routinen oder: Ein Modul für alle Fälle

Auf welcher Ebene man sich auch mit der Basic-Programmierung beschäftigt, so treten bestimmte »Grundprobleme« immer wieder auf. Für den Anfänger kommen diese oft einer unlösbaren Aufgabenstellung gleich, für einen Profi ist es zumindest eine lästige Arbeit. Dies führte zum Begriff »Modulare Programmierung«, womit die Zusammenstellung häufig benötigter Programmteile zwecks mehrfacher Verwendung gemeint ist. Dieses Kapitel beschreibt die wichtigsten Standardmodule in Basic 2.0 und dürfte Ihnen eine große Hilfe bei der Erstellung eigener Programme sein.

3.1 Ein-/Ausgabe

Wie hinlänglich bekannt, läßt sich letztendlich jedes Programm in das Schema »Eingabe – Datenverarbeitung (Berechnungen) – Ausgabe« einfügen. Dieser Abschnitt widmet sich den beiden Eckpfeilern »Eingabe/Ausgabe«, während das darauffolgende Unterkapitel auf die »Datenverarbeitung« eingeht.

3.1.1 PRINT USING – ähnliche Routinen

Die Ausgabe von Zahlen über den herkömmlichen PRINT-Befehl ist für viele Zwecke durchaus ausreichend und zugegebenermaßen einfach zu programmieren. Eine optisch ansprechende – sprich: übersichtliche – Darstellung numerischer Werte, sei es als Tabelle oder als Liste, ist jedoch nur mit Hilfe einer entsprechenden Routine möglich. Folgende Programmzeilen generieren für die Zahl, welche in der Variablen A steht, eine Ausgabe mit Orientierung an der kaufmännischen Zahlendarstellung:

Listing 3.1: KAUFM.DARSTELLG.

```
100 A$=STR$(A)
110 VZ$=LEFT$(A$,1)
120 A$=RIGHT$(A$,LEN(A$)-1)
130 Y=ABS(A)
140 IF Y<1 AND Y>0 THEN A$="0"+A$
150 IF Y-INT(Y)<0.001 THEN A$=A$+".00":GOTO170
160 IF Y*10-INT(Y*10)<0.001 THEN A$=A$+"0"
170 A$=RIGHT$("              "+A$,14)
180 A$=LEFT$(A$,11)+","+RIGHT$(A$,2)
190 IF Y>999.99 THEN A$=LEFT$(A$,8)+"."+RIGHT$(A$,6)
```

```
200 IF Y>999999.99 THEN A$=LEFT$(A$,5)+"."+RIGHT$(A$,10)
210 A$=A$+VZ$
220 PRINT A$
230 RETURN
```

Dies ist ein Beispiel für einen Aufruf:

```
A=15.6:GOSUB 100
```

Dieses Modul ist vorwiegend dafür vorgesehen, Geldbeträge darzustellen. Es dient aber gleichzeitig als Anregung und Basis für eigene Spezialanpassungen. In dieselbe Richtung zielt auch ein Programm zur Eliminierung von Leerzeichen, welche am Ende eines Strings unangenehm ins Auge fallen:

Listing 3.2: ELIMINATOR

```
100 FORI=LEN(NA$)TO1STEP-1
110 IFRIGHT$(NA$,1)=" "THENNA$=LEFT$(NA$,LEN(NA$)-1):NEXT
300 PRINTNA$;",";VN$
```

Beispiel für den Aufruf:

```
NA$="AM ENDE 5 LEERZEICHEN     ":PRINTLEN(NA$),CHR$(34);NA$;CHR$(34)
VN$="WEITER GEHT'S":GOTO 100
```

Bei Strings läßt sich somit der Inhalt in seiner Länge reduzieren. Umgekehrt kommt der übersichtlichen Zahlendarstellung oftmals zugute, wenn eine bestimmte Einheitslänge durch Voranstellung von Nullen erreicht wird; dies ist nicht ohne Grund DIE Zahlendarstellung, die Laien mit dem Begriff »Computer« assoziieren.
Hier nun das Modul, das für die Variable A einen String A$ bildet, welcher anschließend N Zeichen beinhaltet:

Listing 3.3: NULL-ERGAENZUNG

```
100 A$=STR$(A)
110 IFLEN(A$)>=N+1THENPRINT"FEHLER"
120 A$=RIGHT$("0000000000"+MID$(A$,2),N)
```

Nun wieder das Beispiel:

```
A=567:N=5:GOTO 100
```

Danach ist A$=»00567«.

3.1.2 Eingabe-Unterprogramm

Die gravierenden Mängel des INPUT-Befehls in Basic 2.0 sind Ihnen sicher bekannt:

- Eingabe von Semikolon, Komma und Doppelpunkt ist nur in Anführungszeichen möglich
- Eingabe von Anführungszeichen wird ignoriert
- Eingabe läßt sich nicht auf bestimmte Zeichen einschränken (»?REDO FROM START« ist eine Folge dieses Mangels)
- keine Angabe von Minimal- und Maximallänge der Eingabe
- Steuertasten wie <CLR/HOME> oder Cursortasten werden ausgeführt (siehe Kapitel 1)
- vordefinierte Eingabefelder fehlen
- Erweiterbarkeit zu einer Eingabemaske wird vermißt
- freie Programmierbarkeit für maßgeschneiderte Individuallösungen ist unmöglich
- durch <CRSR DOWN> erweitert man versehentlich die logische Eingabezeile auf eine weitere Bildschirmzeile, wogegen kein Kraut – auch nicht <CRSR UP> – mehr gewachsen ist.

Das lästige Fragezeichen, welches vor der Eingabe erscheint, möchte ich schon beiseite lassen, weil es gegen diesen kosmetischen Fehler bereits mehrere trickreiche Mittel gibt.
Doch damit, daß wir das INPUT-Kommando kritisieren, ist es noch lange nicht getan. Sie erhalten nun eine neue, in Basic verfaßte INPUT-Unterroutine von Karlheinz Boss, welche sich durch folgende Vorzüge auszeichnet:

- Da die Routine in Basic geschrieben ist, bleibt sie für jeden Programmierer leicht verständlich und modifizierbar.
- Der Programmierer kann beim Aufruf der Routine gleichzeitig Ort, minimale und maximale Länge und die zulässigen Zeichen bei der Eingabe festlegen.
- Jedes Zeichen wird gleich bei der Eingabe interaktiv auf seine »Richtigkeit« überprüft.
- Es werden selbst bei Fehlbedienungsversuchen nicht Teile der Maske, sondern nur die tatsächliche Eingabe übernommen.
- Sämtliche Steuertasten funktionieren weiterhin, beziehen sich aber speziell auf das jeweilige Eingabefeld und erleichtern damit die Eingabe.
- Der Anwender kann das Eingabefeld beim besten Willen nicht verlassen.
- Die Routine ist mit 1,5 Kbyte relativ kurz und überschaubar.

Zunächst das Demonstrationsprogramm, welches Sie jetzt laden und starten sollten:

Listing 3.4: INPUT (DEMO)

```
0 REM        INPUT (DEMO)
1 REM INPUT-ROUTINE FUR VC 64
2 REM
3 REM COPYRIGHT BY
4 REM KARLHEINZ BOSS
5 REM SYLVESTER-JORDAN-STR. 11
6 REM 3550 MARBURG
7 REM 06421/13509
8 POKE53281,09
  POKE53280,09
  POKE646,07
9 PRINT CHR$(14)
10 REM INPUT-ROUTINE
20 REM
30 REM PA$ IST AUFGEBAUT WIE FOLGT

40 REM PA$ = ZESPMIMAC     WOBEI
50 REM          ZE = ZEILE
60 REM          SP = SPALTE
70 REM          MI = MINDESTLAENGE
80 REM          MA = MAXIMALE LAENGE
90 REM           C = CODE  WOBEI
100 REM            1 = NUR ZAHLEN
110 REM            2 = NUR BUCHSTABEN
120 REM            3 = ALLES
125 REM            4 = NUR KLEINBUCHST.
130 REM          5-6 = FREI PROGRAMMIERBAR
132 REM                ( ZEILE 60490 )
140 REM PA$ MUSS DIE ZEILE UND SPALTE
150 REM BEINHALTEN,DER REST IST HIN-
160 REM REICHEND.ES WERDEN DANN DIE
170 REM STANDARDPARAMETER IN ZEILE
180 REM 60200 ANGENOMMEN
190 REM Z.B. PA$="0510"
200

1000 PRINT"(CLR)";
1010 PRINT"(RVS)  ⌝NPUT-⎽OUTINE   COPYRIGHT BY ⌿.|OSS   (OFF)"
1020 PRINT "⌿EBER GET WIRD EINE ⌝NPUT-⎽OUTINE SIMU-"
1030 PRINT "LIERT, WOBEI SAEMTLICHE ♦ONDERFUNKTIONS-";
1040 PRINT "TASTEN (INST,DEL,HOME,CLR,CR,CL) NOCH "
1050 PRINT "FUNKTIONIEREN, MAN ABER DAS ‾INGABEFELD"
1060 PRINT "NICHT VERLASSEN KANN. ♣USSERDEM KANN MAN";
1070 PRINT "BEIM ♠UFRUF DER ⎽OUTINE GLEICH ┌RT, MIN.";
1080 PRINT "UND MAX. └AENGE UND DIE ZULAESSIGEN ♦EI-";
1090 PRINT "CHEN DER ‾INGABE ANGEBEN."
1100 PRINT "(RVS)(GRY2) |EISP.      ";
1105 PRINT "                                    ";
1110 PRINT "‾ATUM (||⟍⟍⟍⟍)
                    ";

1120 PRINT "                                    ";
1130 PRINT "⌿AME
               ";

1140 PRINT "                                    ";
1150 PRINT "⌿UERZEL (MIND 2 KL. |UCHST.)
          ";

1160 PRINT "                                    ";
1170 PRINT "‾OMPUTER
               ";

1180 PRINT "                          (OFF)(YEL)";
1900 REM HIER NUN DIE 4 AUFRUFE AUS
1910 REM VORANGEGANGEN BEISPIEL
1920 REM
2000 PA$="121806061"
     GOSUB60000
     A$=IN$
```

```
 2010 PA$="140800202"
      GOSUB60000
      B$=IN$
 2020 PA$="163202034"
      GOSUB60000
      C$=IN$
 2030 PA$="1812"
      GOSUB60000
      D$=IN$
 2040 REM
 2100 REM     DAS WAR'S
 2110 REM
 2120 REM PS
       DIE REINE INPUT-ROUTINE
 2130 REM IN KOMPR. FORM HEISST
 2140 REM         INPUT (KURZ)
 3000 PRINT
      PRINT "¯IE ¯INGABEN WAREN
       ";A$
 3010 PRINT B$
 3020 PRINT C$
 3030 PRINT D$
 3040 PRINT "                 (RVS) |ASTE DRUECKEN (OFF)";
 3050 GET NJ$
       IF NJ$="" THEN 3050
 3995 PRINT"(CLR)"
 4000 POKE 211,0
      POKE 214,24
      SYS58640
 4010 PRINT"LIST1900-2140              "
 4015 PRINT"(HOME)"
 4020 LIST30-190
 29999 END
 30000

 31000

 32000

 33000

 34000

 60000 BL$="                                      "
 60010 IN$=""
 60099
 60100 ZE=VAL(MID$(PA$,1,2))
 60110 SP=VAL(MID$(PA$,3,2))
 60120 IF LEN(PA$)=4 THEN 60200
 60130 MI=VAL(MID$(PA$,5,2))
 60140 MA=VAL(MID$(PA$,7,2))
 60150 CO=VAL(MID$(PA$,9,1))
 60160 GOTO 60300
 60190 REM*****************************
 60200 MI=0
       MA=15
       CO=3
 60210 REM*****************************
 60220

 60300 GOSUB 60950
 60310 POKE 204,0
 60320 GET N$
        IF N$="" THEN 60320
 60330 SS=PEEK(211)
```

```
60340 IF ASC(N$)=13 AND LEN(IN$)>=MI THEN POKE 204,1
       GOSUB 63890
      RETURN
60350 IF ASC(N$)=147 THEN GOSUB 61000
       GOTO 60320
60360 IF ASC(N$)=19 THEN GOSUB 63890
       GOSUB 60950
      GOTO 60320
60370 IF ASC(N$)=20 AND PEEK(211)>SP THEN GOSUB 61410
       GOTO 60320
60380 IF ASC(N$)=157 THEN GOSUB 61310
      GOTO 60320
60390 IF ASC(N$)=148 AND LEN(IN$)<MA THEN GOSUB 61505
      GOTO 60320
60400 IF ASC(N$)=29 THEN GOSUB 61210
      GOTO 60320
60410 IF ASC(N$)=17 OR ASC(N$)=145 THEN   60320
60420 IF ASC(N$)=148 OR ASC(N$)=20 OR ASC(N$)=13 THEN GOTO 60320
60430

60485 FL=0
60490 ON CO GOSUB  63900,63910,63920,63930,63940,63950
60495 IF FL=1 GOTO 60320
60500

60505 IF LEN(IN$)=SS-SP THEN IN$=IN$+N$
      GOTO 60515
60510 IN$=LEFT$(IN$,SS-SP)+N$+MID$(IN$,SS-SP+2,LEN(IN$)-SS+SP-1)
60515 PRINT N$;
60520 IF SS=SP+MA-1 THEN PRINT CHR$(157);
60530 GOTO 60320
60901

60902

60903

60950 POKE211,SP
      POKE214,ZE
      SYS58640
      RETURN
60999

61000 GOSUB 60950
61010 PRINT LEFT$(BL$,MA);
61020 GOSUB 60950
61030 IN$=""
61040 RETURN
61099

61210 IF SS<SP+LEN(IN$)AND SS<SP+MA-1THEN SS=SS+1
61220 GOSUB 63890
61230 POKE 211,SS
      SYS58640
61240 RETURN
61250

61310 IF SS>SP THEN SS=SS-1
61320 GOSUB 63890
61330 POKE 211,SS
       SYS 58640
61340 RETURN
61350
```

```
61410 IN$=LEFT$(IN$,SS-SP-1)+MID$(IN$,SS-SP+1,LEN(IN$)-SS+SP)
61420 GOSUB 63890
61425 IF LEN(IN$)<MA-1 THEN PRINT" ";
61430 POKE 211,SS-1
       SYS 58640
61440 RETURN
61490

61505 IF SS=SP+LEN(IN$) THEN GOTO 61540
61510 IN$=LEFT$(IN$,SS-SP)+" "+MID$(IN$,SS-SP+1,LEN(IN$)-SS+SP)
61520 GOSUB 63890
61530 POKE 211,SS
       SYS 58640
61540 RETURN
61590

63890 GOSUB 60950
63892 PRINT IN$;
      IF LEN(IN$)<MA THEN PRINT " ";
63898 RETURN
63899

63900 IF ASC(N$)<48ORASC(N$)>57 THEN FL=1
63909 RETURN
63910 IF ASC(N$)=32 THEN RETURN
63911 IF(ASC(N$)<65ORASC(N$)>90)AND(ASC(N$)<193ORASC(N$)>218) THEN FL=1
63919 RETURN
63920 REM
63929 RETURN
63930 IF ASC(N$)=32 THEN RETURN
63931 IF(ASC(N$)<65ORASC(N$)>90) THEN FL=1
63939 RETURN
63940 REM
63949 RETURN
63950 REM
63959 RETURN
```

Daran lassen sich bereits die Funktionen der Sondertasten austesten:

<CRSR UP> und <CRSR DOWN> sind »kaltgestellt«; wenn Sie die Routine entsprechend für eine eigene Eingabemaske weiterverwenden wollen, bietet sich für <CRSR UP> und <CRSR DOWN> das Wechseln des Eingabefeldes innerhalb einer Eingabemaske an.

<INST> und <DEL> funktionieren wie gewohnt innerhalb des Eingabefeldes, zerstören jedoch niemals den restlichen Bildschirmaufbau.

<HOME> springt an den Anfang des Eingabefenster.

<CLR> löscht das Eingabefenster und springt an seinen Anfang.

<CRSR LEFT> und <CRSR RIGHT> verhalten sich unverändert, verlassen jedoch niemals das Eingabefeld.

Das Demonstrationsprogramm baut in den Zeilen 1000–1180 eine Eingabemaske auf, welche aus den Eingabefeldern »Datum«, »Name«, »Kürzel« und »Computer« besteht. Die Aufrufe der

Eingaberoutine (pro Eingabefeld ein Aufruf) vollziehen sich in 2000-2030. Sie sehen bereits auf den ersten Blick, daß PA$ mit Parametern – Näheres später – belegt, die Routine ab Zeile 60000 aufgerufen und IN$ als Ergebnis zurückgegeben wird. Die Zeilen 3000–29999 geben alle Eingaben noch einmal aus, um damit zu beweisen, daß sie in Strings (A$ = Datum, B$ = Name, C$ = Kürzel, D$ = Computer) gesichert wurden und weiterhin verfügbar bleiben. Nun zur Verwendung der leistungsfähigen Eingaberoutine in eigenen Anwendungen. Dazu sollte man folgende komprimierte Fassung einsetzen, welche ab Zeile 60000 jederzeit »geMERGEt« (an ein bestehendes Programm angehängt) werden kann:

Listing 3.5: INPUT (KURZ)

```
60000 BL$="                                        ":IN$="":ZE=VAL(MID$(PA$,1,2)
):SP=VAL(MID$(PA$,3,2)):IFLEN(PA$)=4THEN60002
60001 MI=VAL(MID$(PA$,5,2)):MA=VAL(MID$(PA$,7,2)):CO=VAL(MID$(PA$,9,1)):GOTO6000
3
60002 MI=0:MA=15:CO=3
60003 GOSUB60019:POKE204,0
60004 GETN$:IFN$=""THEN60004
60005 SS=PEEK(211):IFASC(N$)=13ANDLEN(IN$)>=MITHENPOKE204,1:GOSUB60030:RETURN
60006 IFASC(N$)=147THENGOSUB60020:GOTO60004
60007 IFASC(N$)=19THENGOSUB60030:GOSUB60019:GOTO60004
60008 IFASC(N$)=20ANDPEEK(211)>SPTHENGOSUB60025:GOTO60004
60009 IFASC(N$)=157THENGOSUB60023:GOTO60004
60010 IFASC(N$)=148ANDLEN(IN$)<MATHENGOSUB60027:GOTO60004
60011 IFASC(N$)=29THENGOSUB60021:GOTO60004
60012 IFASC(N$)=17ORASC(N$)=145THEN60004
60013 IFASC(N$)=148ORASC(N$)=20ORASC(N$)=13THENGOTO60004
60014 FL=0:ONCOGOSUB60032,60034,60037,60038,60041,60042:IFFL=1GOTO60004
60015 IFLEN(IN$)=SS-SPTHENIN$=IN$+N$:GOTO60017
60016 IN$=LEFT$(IN$,SS-SP)+N$+MID$(IN$,SS-SP+2,LEN(IN$)-SS+SP-1)
60017 PRINTN$;:IFSS=SP+MA-1THENPRINTCHR$(157);
60018 GOTO60004
60019 POKE211,SP:POKE214,ZE:SYS58640:RETURN
60020 GOSUB60019:PRINTLEFT$(BL$,MA);:GOSUB60019:IN$="":RETURN
60021 IFSS<SP+LEN(IN$)ANDSS<SP+MA-1THENSS=SS+1
60022 GOSUB60030:POKE211,SS:SYS58640:RETURN
60023 IFSS>SPTHENSS=SS-1
60024 GOSUB60030:POKE211,SS:SYS58640:RETURN
60025 IN$=LEFT$(IN$,SS-SP-1)+MID$(IN$,SS-SP+1,LEN(IN$)-SS+SP):GOSUB60030:IFLEN(I
N$)<MA-1THENPRINT" ";
60026 POKE211,SS-1:SYS58640:RETURN
60027 IFSS=SP+LEN(IN$)THENGOTO60029
60028 IN$=LEFT$(IN$,SS-SP)+" "+MID$(IN$,SS-SP+1,LEN(IN$)-SS+SP):GOSUB60030:POKE2
11,SS:SYS58640
60029 RETURN
60030 GOSUB60019:PRINTIN$;:IFLEN(IN$)<MATHENPRINT" ";
60031 RETURN
60032 IFASC(N$)<48ORASC(N$)>57THENFL=1
60033 RETURN
60034 IFASC(N$)=32THENRETURN
60035 IF(ASC(N$)<65ORASC(N$)>90)AND(ASC(N$)<193ORASC(N$)>218)THENFL=1
60036 RETURN
60037 RETURN
60038 IFASC(N$)=32THENRETURN
60039 IF(ASC(N$)<65ORASC(N$)>90)THENFL=1
60040 RETURN
60041 RETURN
60042 RETURN
```

```
60000 BL$="                                        "
      IN$=""
      ZE=VAL(MID$(PA$,1,2))
      SP=VAL(MID$(PA$,3,2))
      IFLEN(PA$)=4THEN60002
60001 MI=VAL(MID$(PA$,5,2))
      MA=VAL(MID$(PA$,7,2))
      CO=VAL(MID$(PA$,9,1))
      GOTO60003
60002 MI=0
      MA=15
      CO=3
60003 GOSUB60019
      POKE204,0
60004 GETN$
      IFN$=""THEN60004
60005 SS=PEEK(211)
      IFASC(N$)=13ANDLEN(IN$)>=MITHENPOKE204,1
      GOSUB60030
      RETURN
60006 IFASC(N$)=147THENGOSUB60020
      GOTO60004
60007 IFASC(N$)=19THENGOSUB60030
      GOSUB60019
      GOTO60004
60008 IFASC(N$)=20ANDPEEK(211)>SPTHENGOSUB60025
      GOTO60004
60009 IFASC(N$)=157THENGOSUB60023
      GOTO60004
60010 IFASC(N$)=148ANDLEN(IN$)<MATHENGOSUB60027
      GOTO60004
60011 IFASC(N$)=29THENGOSUB60021
      GOTO60004
60012 IFASC(N$)=17ORASC(N$)=145THEN60004
60013 IFASC(N$)=148ORASC(N$)=20ORASC(N$)=13THENGOTO60004
60014 FL=0
      ONCOGOSUB60032,60034,60037,60038,60041,60042
      IFFL=1GOTO60004
60015 IFLEN(IN$)=SS-SPTHENIN$=IN$+N$
      GOTO60017
60016 IN$=LEFT$(IN$,SS-SP)+N$+MID$(IN$,SS-SP+2,LEN(IN$)-SS+SP-1)
60017 PRINTN$;
      IFSS=SP+MA-1THENPRINTCHR$(157);
60018 GOTO60004
60019 POKE211,SP
      POKE214,ZE
      SYS58640
      RETURN
60020 GOSUB60019
      PRINTLEFT$(BL$,MA);
      GOSUB60019
      IN$=""
      RETURN
60021 IFSS<SP+LEN(IN$)ANDSS<SP+MA-1THENSS=SS+1
60022 GOSUB60030
      POKE211,SS
      SYS58640
      RETURN
60023 IFSS>SPTHENSS=SS-1
60024 GOSUB60030
      POKE211,SS
      SYS58640
      RETURN
60025 IN$=LEFT$(IN$,SS-SP-1)+MID$(IN$,SS-SP+1,LEN(IN$)-SS+SP)
      GOSUB60030
      IFLEN(IN$)<MA-1THENPRINT" ";
```

```
60026 POKE211,SS-1
      SYS58640
      RETURN
60027 IFSS=SP+LEN(IN$)THENGOTO60029
60028 IN$=LEFT$(IN$,SS-SP)+" "+MID$(IN$,SS-SP+1,LEN(IN$)-SS+SP)
      GOSUB60030
      POKE211,SS
      SYS58640
60029 RETURN
60030 GOSUB60019
      PRINTIN$;
      IFLEN(IN$)<MATHENPRINT" ";
60031 RETURN
60032 IFASC(N$)<48ORASC(N$)>57THENFL=1
60033 RETURN
60034 IFASC(N$)=32THENRETURN
60035 IF(ASC(N$)<65ORASC(N$)>90)AND(ASC(N$)<193ORASC(N$)>218)THENFL=1
60036 RETURN
60037 RETURN
60038 IFASC(N$)=32THENRETURN
60039 IF(ASC(N$)<65ORASC(N$)>90)THENFL=1
60040 RETURN
60041 RETURN
60042 RETURN
```

Die prinzipielle, in drei Schritte unterteilte Nutzung der Routine ist Ihnen bereits vom Demoprogramm bekannt:

a) Setzen des Parameterstrings PA$
b) Aufruf mit GOSUB 60000
c) Sichern von IN$ durch Übertragung in andere Variable

Wichtig ist nun der Aufbau von PA$, unserem »Wunschzettel« an die Eingaberoutine:

Stelle		Bedeutung
Stelle 1,2	:	Zeile der Eingabe
Stelle 3,4	:	Spalte der Eingabe
Stelle 5,6	:	minimale Länge der Eingabe
Stelle 7,8	:	maximale Länge
Stelle 9	:	Code für erlaubte Eingabezeichen 1 = nur Zahlen 2 = nur Buchstaben 3 = alle Zeichen 4 = nur Kleinbuchstaben

Es ist hierbei zu beachten, daß einstellige Angaben (zum Beispiel: »3« für 3. Zeile) auf zwei Stellen aufgefüllt werden müssen (»03«), weil ansonsten keine klare Trennung der einzelnen Parameter möglich wäre. Die Routine sieht Standardwerte für minimale und maximale Länge sowie den Eingabecode vor (Zeile 60002), welche natürlich frei nach eigenem Bedarf eingerichtet werden können. Entspricht die geforderte Eingabe den »Defaults«, so braucht der Programmierer beim Setzen des Parameterstrings diese nicht anzugeben, sondern nur noch Zeile und Spalte (Beispiel: pa$=»0510«, d.h. Eingabe in Zeile 5 ab Spalte 10 mit den Default-Werten). Dies führt, wie Sie sehen, zu einer weiteren Vereinfachung, da bereits

vier Zeichen als PA$-Inhalt genügen. Merken Sie sich als Regel: PA$ enthält entweder alle 9 Stellen oder in verkürzter Form (mit Defaults) genau 4 Stellen; Zwischenwerte finden keine Berücksichtigung.
Abschließend noch eine Bemerkung zum »Code«. Dieser dient zur sofortigen Überprüfung eingegebener Zeichen auf deren Richtigkeit. Selbstverständlich kann er auch frei programmiert werden: Zeile 60014 verzweigt dem Code CO entsprechend in die einzelnen Unterprogramme:

CO=1	:	Zeilen 60032/60033
CO=2	:	Zeilen 60034–60036
CO=3	:	Zeile 60037
CO=4	:	Zeilen 60038–60040
CO=5	:	Zeile 60041 (momentan ungenutzt)
CO=6	:	Zeile 60042 (momentan ungenutzt)

Diese Sonderbehandlungen prüfen, ob angesichts des gewählten Eingabecodes das in N$ enthaltene Zeichen übernommen werden darf; falls ja, ist FL=1 zu setzen, ansonsten wird nichts unternommen (in 60014 bekommt FL den Ausgangswert 0 für »unerlaubtes Zeichen« zugewiesen). Die Codes 5 und 6 entsprechen momentan Code 60037, sind aber eigentlich zur eigenen Modifikation vorgesehen.

3.1.3 Diskettenbehandlung

Für ein wirklich gutes Anwendungsprogramm sollte ein Disketten-Hilfsmenü Ehrensache sein; die Bemühungen um eine entsprechende Option scheitern jedoch in Basic fast immer am Problem der Directory-Ausgabe. Entweder sind die bislang veröffentlichten Routinen bei weitem nicht schnell genug oder sie funktionieren nur teilweise, und wenn einmal keiner dieser beiden Mängel vorliegt, so handelt es sich um ein Maschinenprogramm.
Doch sehen Sie sich einmal folgendes Programm an:

Listing 3.6: DIRECTORY

```
1 PRINT"(CLR)"
  OPEN1,8,0,"$"
  POKE781,1
  SYS65478
  GETA$,A$
  E$=CHR$(0)
2 GETA$,A$,H$,L$
  IFSTTHENSYS65484
  CLOSE1
  END
3 PRINTASC(H$+E$)+256*ASC(L$+E$);
4 GETA$,B$
  IFA$THENPRINTA$B$;
  GOTO4
5 PRINTA$
  GOTO2
```

```
1 PRINT"(CLR)":OPEN1,8,0,"$":POKE781,1:SYS65478:GETA$,A$:E$=CHR$(0)
2 GETA$,A$,H$,L$:IFSTTHENSYS65484:CLOSE1:END
3 PRINTASC(H$+E$)+256*ASC(L$+E$);
4 GETA$,B$:IFA$THENPRINTA$B$;:GOTO4
5 PRINTA$:GOTO2
```

Es gibt ein Directory mit – für Basic – atemberaubender Geschwindigkeit im selben Format aus, wie man es von LOAD "$",8 kennt. Bei Anpassung der GOTOs in den Zeilen 4 und 5 läßt sich dieses Modul tausendfach einsetzen.
Den Kern der Routine bilden die Kernel-Aufrufe SYS65478 und SYS65484 (siehe »C64 für Insider«).
Ein komplettes Diskettenmenü ist folgendes Programm:

Listing 3.7: DISK-MENUE

```
1 PRINT"(CLR)DEMO DES DISKETTEN-HILFSMENUES
  "
2 PRINT"(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)
(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)";
  GOSUB50000
  END
3

4

5

50000 PRINT"(RVS)┌────────────────┐(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEF
T)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50010 PRINT"|DISKETTEN-MENUE|(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50020 PRINT"├────────────────┤(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50030 PRINT"|<D> DIRECTORY   |(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50040 PRINT"|<B> BEFEHL      |(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50050 PRINT"|<S> STATUS      |(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50060 PRINT"|<Z> ZURUECK     |(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50070 PRINT"└────────────────┘(DOWN)(DOWN)(UP)(UP)"
50080 GET DM$
      IF DM$<>"" THEN 50080
50090 GET DM$
      IF DM$<>"D" AND DM$<>"B" AND DM$<>"S" AND DM$<>"Z" THEN 50090
50100 IF DM$="Z" THEN RETURN
50110 IFDM$<>"B"THEN50150
50120 SYS42336
      XX$=""
      II=512
50130 AA=PEEK(II)
      IF AA THEN XX$=XX$+CHR$(AA)
      II=II+1
      GOTO50130
```

```
50140 OPEN1,8,15,XX$
      CLOSE1
      PRINT"(UP)";
      DM$="S"
50150 IFDM$="S"THENOPEN1,8,15
      INPUT#1,DM,DM$,D1,D2
      PRINT"(DOWN)"DM;DM$;D1;D2;"(UP)(UP)"
      CLOSE1
50160 IF DM$<>"D" THEN 50080
50170 PRINT"(CLR)"
      OPEN1,8,0,"$"
      POKE781,1
      SYS65478
      GETA$,A$
      E$=CHR$(0)
50180 GETA$,A$,H$,L$
      IFSTTHENSYS65484
      CLOSE1
      GOTO50220
50190 PRINTASC(H$+E$)+256*ASC(L$+E$);
50200 GETA$,B$
      IFA$THENPRINTA$B$;
      GOTO50200
50210 PRINTA$
      GOTO50180
50220 GETA$
      IFA$<>CHR$(13)THEN50220
50230 PRINT"(CLR)";
      GOTO50000
```

```
1 PRINT"(CLR)DEMO DES DISKETTEN-HILFSMENUES:"
2 PRINT"(DOWN)(DOWN)(DOWN)(DOWN)(DOWN)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)
(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)(RGHT)";:GOSUB
50000:END
3 :
4 :
5 :
50000 PRINT"(RVS)┌───────────────┐(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEF
T)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50010 PRINT"|DISKETTEN-MENUE|(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50020 PRINT"├───────────────┤(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50030 PRINT"|<D> DIRECTORY  |(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50040 PRINT"|<B> BEFEHL     |(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50050 PRINT"|<S> STATUS     |(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50060 PRINT"|<Z> ZURUECK    |(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LE
FT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(LEFT)(DOWN)";
50070 PRINT"└───────────────┘(DOWN)(DOWN)(UP)(UP)"
50080 GET DM$:IF DM$<>"" THEN 50080
50090 GET DM$:IF DM$<>"D" AND DM$<>"B" AND DM$<>"S" AND DM$<>"Z" THEN 50090
50100 IF DM$="Z" THEN RETURN
50110 IFDM$<>"B"THEN50150
50120 SYS42336:XX$="":II=512
50130 AA=PEEK(II):IF AA THEN XX$=XX$+CHR$(AA):II=II+1:GOTO50130
50140 OPEN1,8,15,XX$:CLOSE1:PRINT"(UP)";:DM$="S"
50150 IFDM$="S"THENOPEN1,8,15:INPUT#1,DM,DM$,D1,D2:PRINT"(DOWN)"DM;DM$;D1;D2;"(U
P)(UP)":CLOSE1
```

```
50160 IF DM$<>"D" THEN 50080
50170 PRINT"(CLR)":OPEN1,8,0,"$":POKE781,1:SYS65478:GETA$,A$:E$=CHR$(0)
50180 GETA$,A$,H$,L$:IFSTTHENSYS65484:CLOSE1:GOTO50220
50190 PRINTASC(H$+E$)+256*ASC(L$+E$);
50200 GETA$,B$:IFA$THENPRINTA$B$;:GOTO50200
50210 PRINTA$:GOTO50180
50220 GETA$:IFA$<>CHR$(13)THEN50220
50230 PRINT"(CLR)";:GOTO50000
```

Die große Besonderheit liegt eindeutig darin, daß das Disk-Menü zunächst an der aktuellen Cursorposition aufgebaut ist; bei Verwendung der Directory-Anzeige läßt es sich jedoch nicht vermeiden, daß der gesamte Bildschirmaufbau gelöscht werden muß. Ab Zeile 50 170 finden Sie sicher die Directory-Routine aus Listing 3.6 wieder. Die Eingaberoutine in 50 120/50 130 ist eine Verbesserung des entsprechenden Einzeilers #16, welche etwas sauberer in zwei Zeilen programmiert wurde; der Fortschritt besteht darin, daß die II-Schleife nicht mehr unvermittelt abgebrochen wird, sondern nun ordnungsgemäß abläuft.
Das Diskettenmenü-Programm steht ab Zeile 50000 und wird Ihnen sicher noch viel Freude bereiten, wenn Sie Ihre Programme damit aufwerten.

3.2 Datenverarbeitung

Haben Sie schon immer wissen wollen, wie man blitzschnell Strings und Zahlen sortiert? Möchten Sie Variablenfelder nach dem Zufallsprinzip mischen? Fehlen Ihnen bestimmte Stringoperationen in Basic 2.0? Wollen Sie »up to date« sein? Oder liegt Ihnen an der unproblematischen Umrechnung zwischen den verschiedensten Zahlensystemen?
Dann erhalten Sie hier maßgeschneiderte Fertiglösungen.

3.2.1 Sortieren mit Mischsort

Im Dschungel der Sortieralgorithmen ist es schwer, genau den richtigen zu finden. Hier möchte ich nur denjenigen vorstellen, welchen ich selbst aus vielerlei Gründen allen anderen vorziehe:

Listing 3.8: MISCHSORT

```
10 REM ERSTELLEN EINES FELDES ZUM
20 REM SORTIEREN.
30 REM DAS ERSTELLEN KANN ZUFAELLIG
40 REM ODER GEZIELT (DURCH EINGABE)
50 REM ERFOLGEN.
60 REM
70 REM SORTIERALGORITHMEN ERHALTEN DIE
80 REM ZEILENNUMMERN VON 10000 BIS 50000
```

```
90 REM SIE BENOETIGEN JEWEILS DIESEN
99 REM VORSPANN ZUR AUSFUEHRUNG.
100 REM HERSTELLUNG EINES ARRAYS

110 REM ARRAYVARIABLE        - A$
120 REM SCHLEIFENVARIABLEN - X, Y, Z
130 REM HILFSVARIABLEN     - B$, C$, D$
140 REM DREIECKTAUSCH MIT  - S$
150 PRINT"(CLR)"
    CLR
160 PRINT"SOLL VON (RVS)H(OFF)AND ODER (RVS)Z(OFF)UFAELLIG ERSTELLT"
    PRINT
170 INPUT"WERDEN ";X$
180 IFX$<>"H"ANDX$<>"Z"THEN150
190 IFX$="H"THENGOSUB220
    GOSUB1000
    GOTO210
200 GOSUB220
    GOSUB2000
210 GOTO4000
     REM WEITERMACHEN
220 REM ANZAHL DER ELEMENTE BESTIMMEN
230 PRINT
    INPUT"ANZAHL DER ELEMENTE ";A
240 IF A>10000THENPRINT
    PRINT"ZU VIELE ELEMENTE"
    GOTO230
250 IFA<10THENPRINT
    PRINT"ZU WENIGE ELEMENTE"
    GOTO230
255 DIM A$(A)
260 INPUT"(DOWN)(DOWN)(RVS)D(OFF)RUCKER ODER (RVS)B(OFF)ILDSCHIRM ";Y$
270 IFY$<>"D"ANDY$<>"B"THEN260
280 IFY$="D"THEND=4
    GOTO300
290 D=3
300 RETURN
1000 REM EINGABE VON HAND
1010 PRINT"(CLR)VORWAERTS ODER RUECKWAERTS";
1020 INPUTX$
     IFX$<>"R"ANDX$<>"V"THEN1020
1030 R=1
     X1=1
     X2=A
1040 IFX$="R"THENR=-1
     X1=A
     X2=1
1050 Y=0
     FORX=X1TOX2STEPR
1060 Y=Y+1
     A$(Y)=STR$(X+99)
1070 NEXTX
1080 RETURN
2000 REM ZUFAELLIGE EINGABE
2010 PRINT"(CLR)"
2020 PRINT
     PRINT"ES WERDEN JETZT"A" ELEMENTE ZUFAELLIG"
     PRINT
     PRINT"AUSGEWAEHLT"
2030 PRINT
     PRINT"JEDES ELEMENT BESTEHT AUS 3 ZEICHEN."
     PRINT
     PRINT
2040 FORX=1TOA
2050 A$(X)=""
```

```
2060 FORY=1TO3
     A$(X)=A$(X)+CHR$(INT(RND(TI)*25)+65)
     NEXTY
2070 NEXTX
2080 RETURN
3000 REM ZWISCHENAUSGABE DER ELEMENTE
3010 FOR I=1TOA-9STEP10
3020 FOR J=ITOI+9
     PRINT#1,A$(J)" ";
     NEXTJ
3030 PRINT#1
     NEXTI
3040 RETURN
4000 REM WEITERMACHEN
4005 OPEN1,D
4010 PRINT"(CLR)AUSGABE DES ERSTELLTEN FELDES"
4020 PRINT
4030 GOSUB3000
4040 REM SORTIERUNG STARTET
4050 REM
10040 TI$="000000"
10050 ZF=INT(A/2+.5)
      DIMTV$(ZF)
10060 FORX=2TOASTEP2
10070 IFA$(X-1)>A$(X)THENTV$(0)=A$(X-1)
      A$(X-1)=A$(X)
      A$(X)=TV$(0)
10080 NEXTX
      IFA<3THEN 50000
10090 LF=1
      FORMZ=1TOINT(LOG(A-1)/LOG(2))
10100 ZF=INT(ZF/2+.5)
      LF=2*LF
10110 FORFZ=1TOZF
      A1=1+2*LF*(FZ-1)
      A2=A1+LF
      E1=A2-1
      E2=E1+LF
10120 IFE2<=ATHENET=LF
      GOTO10150
10130 IFA<A2THEN10230
10140 E2=A
      ET=A+1-A2
10150 FORX=1TOET
      TV$(X)=A$(X+E1)
      NEXT
10160 FORX=E2TOA1STEP-1
10170 IFA$(E1)<TV$(ET)THEN10200
10180 A$(X)=A$(E1)
      E1=E1-1
      IFE1>=A1THEN10220
10190 FORY=X-1TOA1STEP-1
      A$(Y)=TV$(ET)
      ET=ET-1
      NEXTY
      GOTO10210
10200 A$(X)=TV$(ET)
      ET=ET-1
      IFET>=1THEN10220
10210 X=A1
10220 NEXTX
10230 NEXTFZ,MZ
50000 T$=TI$
      REM ENDEBEHANDLUNG
```

```
50010 PRINT#1
50020 GOSUB 3000
50030 PRINT#1,A;" ELEMENTE"
50040 PRINT#1
      PRINT#1
      PRINT#1,T$
      CLOSE1
50050 END
```

```
10 REM ERSTELLEN EINES FELDES ZUM
20 REM SORTIEREN.
30 REM DAS ERSTELLEN KANN ZUFAELLIG
40 REM ODER GEZIELT (DURCH EINGABE)
50 REM ERFOLGEN.
60 REM
70 REM SORTIERALGORITHMEN ERHALTEN DIE
80 REM ZEILENNUMMERN VON 10000 BIS 50000
90 REM SIE BENOETIGEN JEWEILS DIESEN
99 REM VORSPANN ZUR AUSFUEHRUNG.
100 REM HERSTELLUNG EINES ARRAYS:
110 REM ARRAYVARIABLE       - A$
120 REM SCHLEIFENVARIABLEN - X, Y, Z
130 REM HILFSVARIABLEN      - B$, C$, D$
140 REM DREIECKTAUSCH MIT   - S$
150 PRINT"(CLR)":CLR
160 PRINT"SOLL VON (RVS)H(OFF)AND ODER (RVS)Z(OFF)UFAELLIG ERSTELLT":PRINT
170 INPUT"WERDEN ";X$
180 IFX$<>"H"ANDX$<>"Z"THEN150
190 IFX$="H"THENGOSUB220:GOSUB1000:GOTO210
200 GOSUB220:GOSUB2000
210 GOTO4000: REM WEITERMACHEN
220 REM ANZAHL DER ELEMENTE BESTIMMEN
230 PRINT:INPUT"ANZAHL DER ELEMENTE ";A
240 IF A>10000THENPRINT:PRINT"ZU VIELE ELEMENTE":GOTO230
250 IFA<10THENPRINT:PRINT"ZU WENIGE ELEMENTE":GOTO230
255 DIM A$(A)
260 INPUT"(DOWN)(DOWN)(RVS)D(OFF)RUCKER ODER (RVS)B(OFF)ILDSCHIRM ";Y$
270 IFY$<>"D"ANDY$<>"B"THEN260
280 IFY$="D"THEND=4:GOTO300
290 D=3
300 RETURN
1000 REM EINGABE VON HAND
1010 PRINT"(CLR)VORWAERTS ODER RUECKWAERTS";
1020 INPUTX$:IFX$<>"R"ANDX$<>"V"THEN1020
1030 R=1:X1=1:X2=A
1040 IFX$="R"THENR=-1:X1=A:X2=1
1050 Y=0:FORX=X1TOX2STEPR
1060 Y=Y+1:A$(Y)=STR$(X+99)
1070 NEXTX
1080 RETURN
2000 REM ZUFAELLIGE EINGABE
2010 PRINT"(CLR)"
2020 PRINT:PRINT"ES WERDEN JETZT"A" ELEMENTE ZUFAELLIG":PRINT:PRINT"AUSGEWAEHLT"
2030 PRINT:PRINT"JEDES ELEMENT BESTEHT AUS 3 ZEICHEN.":PRINT:PRINT
2040 FORX=1TOA
2050 A$(X)=""
2060 FORY=1TO3:A$(X)=A$(X)+CHR$(INT(RND(TI)*25)+65):NEXTY
2070 NEXTX
2080 RETURN
3000 REM ZWISCHENAUSGABE DER ELEMENTE
3010 FOR I=1TOA-9STEP10
```

```
3020 FOR J=ITOI+9:PRINT#1,A$(J)" ";:NEXTJ
3030 PRINT#1:NEXTI
3040 RETURN
4000 REM WEITERMACHEN
4005 OPEN1,D
4010 PRINT"(CLR)AUSGABE DES ERSTELLTEN FELDES"
4020 PRINT
4030 GOSUB3000
4040 REM SORTIERUNG STARTET
4050 REM
10040 TI$="000000"
10050 ZF=INT(A/2+.5):DIMTV$(ZF)
10060 FORX=2TOASTEP2
10070 IFA$(X-1)>A$(X)THENTV$(0)=A$(X-1):A$(X-1)=A$(X):A$(X)=TV$(0)
10080 NEXTX:IFA<3THEN 50000
10090 LF=1:FORMZ=1TOINT(LOG(A-1)/LOG(2))
10100 ZF=INT(ZF/2+.5):LF=2*LF
10110 FORFZ=1TOZF:A1=1+2*LF*(FZ-1):A2=A1+LF:E1=A2-1:E2=E1+LF
10120 IFE2<=ATHENET=LF:GOTO10150
10130 IFA<A2THEN10230
10140 E2=A:ET=A+1-A2
10150 FORX=1TOET:TV$(X)=A$(X+E1):NEXT
10160 FORX=E2TOA1STEP-1
10170 IFA$(E1)<TV$(ET)THEN10200
10180 A$(X)=A$(E1):E1=E1-1:IFE1>=A1THEN10220
10190 FORY=X-1TOA1STEP-1:A$(Y)=TV$(ET):ET=ET-1:NEXTY:GOTO10210
10200 A$(X)=TV$(ET):ET=ET-1:IFET>=1THEN10220
10210 X=A1
10220 NEXTX
10230 NEXTFZ,MZ
50000 T$=TI$:REM ENDEBEHANDLUNG
50010 PRINT#1
50020 GOSUB 3000
50030 PRINT#1,A;" ELEMENTE"
50040 PRINT#1:PRINT#1:PRINT#1,T$:CLOSE1
50050 END
```

Hierbei enthalten die Zeilen 10050–10230 den eigentlichen Sortieralgorithmus; alles andere ist ein Rahmenprogramm, welches die Erstellung eines Testfeldes komfortabel ermöglicht und an die Sortierroutine folgende Parameter übermittelt:

A$() In diesem Array stehen die zu sortierenden Strings.
A Dies ist die Anzahl der sortierenden Elemente von A$.

Folgende Variablen verwendet Mischsort selbst:

```
A1, A2, E1, E2, ET, FZ, LF, MZ, TV$(), X, Y, ZF
```

Das Prinzip dieses Sortieralgorithmus ist folgendes: Ähnlich wie beim berühmten Quicksort wird das zu sortierende Array A$() in Untereinheiten (Teilfelder) zergliedert; diese Teilfelder fangen bei der Größe 2 an und nehmen dann im Quadrat an Umfang zu (4, 8, 16, ...). Zuerst werden die Untereinheiten der Größe 2 jeweils für sich sortiert, so daß wir A/2 sortierte Teilfelder haben (A ist die Anzahl der Elemente im Gesamtfeld). Jeweils zwei benachbarte Teilfelder werden nun »zusammengemischt« und wiederum sortiert. Setzt man dies über das gesamte

Array hinweg fort, so ergibt dies A/4 sortierte Teilfelder der Länge 4. Als nächstes werden jeweils zwei benachbarte Teilfelder der Länge 4 zusammengemischt und sortiert; dies wird solange wiederholt, bis schließlich nur noch ein (Teil-)Feld übrigbleibt, nämlich das rundum sortierte Gesamtfeld.
Nun zum Gesamturteil über Mischsort: Es erreicht traumhafte Zeiten für Basic-Sortierprogramme, welche teilweise nur 50 Prozent der Zeiten eines adäquaten Quicksort betragen. Es wird auch klar, daß bei Mischsort die »Garbage Collection« (siehe »C64 für Insider«) deutlich später als bei Quicksort einsetzt, was aber wahrlich nicht der einzige Grund für die überragende Geschwindigkeit ist. Mischsort ist genauso einsatzfähig wie Quicksort (ohne »Macken«) und kann deshalb als bestes Sortierprogramm in Basic betrachtet werden. Haben Sie viele große Felder zu sortieren, so spielt es bei diesem Algorithmus fast keine Rolle, ob die Felder vorsortiert sind oder nicht; schnell ist er auf jeden Fall!

3.2.2 Mischen

Trug das Programm »Mischsort« aus 3.2.1 diesen Titel, weil es aus mehreren Sortieralgorithmen »gemischt« wurde, so soll nun das eigentliche Mischen – also zufallsabhängiges Sortieren von Feldern – realisiert werden:

Listing 3.9: MISCHEN

```
10 REM ERSTELLEN EINES FELDES ZUM
20 REM SORTIEREN.
30 REM DAS ERSTELLEN KANN ZUFAELLIG
40 REM ODER GEZIELT (DURCH EINGABE)
50 REM ERFOLGEN.
60 REM
70 REM SORTIERALGORITHMEN ERHALTEN DIE
80 REM ZEILENNUMMERN VON 10000 BIS 50000
90 REM SIE BENOETIGEN JEWEILS DIESEN
99 REM VORSPANN ZUR AUSFUEHRUNG.
100 REM HERSTELLUNG EINES ARRAYS

110 REM ARRAYVARIABLE      - A$
120 REM SCHLEIFENVARIABLEN - X, Y, Z
130 REM HILFSVARIABLEN     - B$, C$, D$
140 REM DREIECKTAUSCH MIT  - S$
150 PRINT"(CLR)"
    CLR
160 PRINT"SOLL VON (RVS)H(OFF)AND ODER (RVS)Z(OFF)UFAELLIG ERSTELLT"
    PRINT
170 INPUT"WERDEN ";X$
180 IFX$<>"H"ANDX$<>"Z"THEN150
190 IFX$="H"THENGOSUB220
    GOSUB1000
    GOTO210
200 GOSUB220
    GOSUB2000
210 GOTO4000
    REM WEITERMACHEN
220 REM ANZAHL DER ELEMENTE BESTIMMEN
230 PRINT
    INPUT"ANZAHL DER ELEMENTE ";A
```

```
240 IF A>10000THENPRINT
    PRINT"ZU VIELE ELEMENTE"
    GOTO230
250 IFA<10THENPRINT
    PRINT"ZU WENIGE ELEMENTE"
    GOTO230
255 DIM A$(A)
260 INPUT"(DOWN)(DOWN)(RVS)D(OFF)RUCKER ODER (RVS)B(OFF)ILDSCHIRM ";Y$
270 IFY$<>"D"ANDY$<>"B"THEN260
280 IFY$="D"THEND=4
    GOTO300
290 D=3
300 RETURN
1000 REM EINGABE VON HAND
1010 PRINT"(CLR)VORWAERTS ODER RUECKWAERTS";
1020 INPUTX$
     IFX$<>"R"ANDX$<>"V"THEN1020
1030 R=1
     X1=1
     X2=A
1040 IFX$="R"THENR=-1
     X1=A
     X2=1
1050 Y=0
     FORX=X1TOX2STEPR
1060 Y=Y+1
     A$(Y)=STR$(X+99)
1070 NEXTX
1080 RETURN
2000 REM ZUFAELLIGE EINGABE
2010 PRINT"(CLR)"
2020 PRINT
     PRINT"ES WERDEN JETZT"A"ELEMENTE ZUFAELLIG"
     PRINT
     PRINT"AUSGEWAEHLT."
2030 PRINT
     PRINT"JEDES ELEMENT BESTEHT AUS 3 ZEICHEN."
     PRINT
     PRINT
2040 FORX=1TOA
2050 A$(X)=""
2060 FORY=1TO3
     A$(X)=A$(X)+CHR$(INT(RND(TI)*25)+65)
     NEXTY
2070 NEXTX
2080 RETURN
3000 REM ZWISCHENAUSGABE DER ELEMENTE
3010 FOR I=1TOA-9STEP10
3020 FOR J=ITOI+9
     PRINT#1,A$(J)" ";
     NEXTJ
3030 PRINT#1
     NEXTI
3040 RETURN
4000 REM WEITERMACHEN
4005 OPEN1,D
4010 PRINT"(CLR)AUSGABE DES ERSTELLTEN FELDES"
4020 PRINT
4030 GOSUB3000
4040 REM SORTIERUNG STARTET
4050 REM
10000 I=RND(-TI)
      TI$="000000"
10010 FOR I=1 TO A
10020 FL=INT(RND(TI)*A)+1
```

```
10030 S$=A$(I)
      A$(I)=A$(FL)
      A$(FL)=S$
10040 NEXT
50000 T$=TI$
      REM ENDEBEHANDLUNG
50010 PRINT#1
50020 GOSUB 3000
50030 PRINT#1,A;" ELEMENTE"
50040 PRINT#1
      PRINT#1
      PRINT#1,T$
      CLOSE1
50050 END
```

```
10 REM ERSTELLEN EINES FELDES ZUM
20 REM SORTIEREN.
30 REM DAS ERSTELLEN KANN ZUFAELLIG
40 REM ODER GEZIELT (DURCH EINGABE)
50 REM ERFOLGEN.
60 REM
70 REM SORTIERALGORITHMEN ERHALTEN DIE
80 REM ZEILENNUMMERN VON 10000 BIS 50000
90 REM SIE BENOETIGEN JEWEILS DIESEN
99 REM VORSPANN ZUR AUSFUEHRUNG.
100 REM HERSTELLUNG EINES ARRAYS:
110 REM ARRAYVARIABLE       - A$
120 REM SCHLEIFENVARIABLEN - X, Y, Z
130 REM HILFSVARIABLEN      - B$, C$, D$
140 REM DREIECKTAUSCH MIT  - S$
150 PRINT"(CLR)":CLR
160 PRINT"SOLL VON (RVS)H(OFF)AND ODER (RVS)Z(OFF)UFAELLIG ERSTELLT":PRINT
170 INPUT"WERDEN ";X$
180 IFX$<>"H"ANDX$<>"Z"THEN150
190 IFX$="H"THENGOSUB220:GOSUB1000:GOTO210
200 GOSUB220:GOSUB2000
210 GOTO4000: REM WEITERMACHEN
220 REM ANZAHL DER ELEMENTE BESTIMMEN
230 PRINT:INPUT"ANZAHL DER ELEMENTE ";A
240 IF A>10000THENPRINT:PRINT"ZU VIELE ELEMENTE":GOTO230
250 IFA<10THENPRINT:PRINT"ZU WENIGE ELEMENTE":GOTO230
255 DIM A$(A)
260 INPUT"(DOWN)(DOWN)(RVS)D(OFF)RUCKER ODER (RVS)B(OFF)ILDSCHIRM ";Y$
270 IFY$<>"D"ANDY$<>"B"THEN260
280 IFY$="D"THEND=4:GOTO300
290 D=3
300 RETURN
1000 REM EINGABE VON HAND
1010 PRINT"(CLR)VORWAERTS ODER RUECKWAERTS";
1020 INPUTX$:IFX$<>"R"ANDX$<>"V"THEN1020
1030 R=1:X1=1:X2=A
1040 IFX$="R"THENR=-1:X1=A:X2=1
1050 Y=0:FORX=X1TOX2STEPR
1060 Y=Y+1:A$(Y)=STR$(X+99)
1070 NEXTX
1080 RETURN
2000 REM ZUFAELLIGE EINGABE
2010 PRINT"(CLR)"
2020 PRINT:PRINT"ES WERDEN JETZT"A"ELEMENTE ZUFAELLIG":PRINT:PRINT"AUSGEWAEHLT."
2030 PRINT:PRINT"JEDES ELEMENT BESTEHT AUS 3 ZEICHEN.":PRINT:PRINT
2040 FORX=1TOA
2050 A$(X)=""
```

```
2060 FORY=1TO3:A$(X)=A$(X)+CHR$(INT(RND(TI)*25)+65):NEXTY
2070 NEXTX
2080 RETURN
3000 REM ZWISCHENAUSGABE DER ELEMENTE
3010 FOR I=1TOA-9STEP10
3020 FOR J=ITOI+9:PRINT#1,A$(J)" ";:NEXTJ
3030 PRINT#1:NEXTI
3040 RETURN
4000 REM WEITERMACHEN
4005 OPEN1,D
4010 PRINT"(CLR)AUSGABE DES ERSTELLTEN FELDES"
4020 PRINT
4030 GOSUB3000
4040 REM SORTIERUNG STARTET
4050 REM
10000 I=RND(-TI):TI$="000000"
10010 FOR I=1 TO A
10020 FL=INT(RND(TI)*A)+1
10030 S$=A$(I):A$(I)=A$(FL):A$(FL)=S$
10040 NEXT
50000 T$=TI$:REM ENDEBEHANDLUNG
50010 PRINT#1
50020 GOSUB 3000
50030 PRINT#1,A;" ELEMENTE"
50040 PRINT#1:PRINT#1:PRINT#1,T$:CLOSE1
50050 END
```

Eingebunden in dasselbe Rahmenprogramm wie Mischsort, steht das eigentliche Modul in den Zeilen 10000–10040, wobei die Anweisung TI=»000000« in Zeile 10000 nicht zum Funktionieren des Mischvorgangs, sondern ausschließlich zur Zeitmessung erforderlich ist. Die Initialisierung des Zufallsgenerators, wie sie in derselben Programmzeile durch I=RND(-TI) geschieht, ist ein absolutes Muß, ansonsten würde das Mischen oftmals nicht »zufällig« genug sein. Verwendet werden die Variablen I als Schleifenzähler, FL als Index für ein auszutauschendes Element und S$ als Hilfsvariable beim Stringtausch.

3.2.3 Datumsauswertung

In vielen Bereichen der Datenverarbeitung kommt dem Datum besondere Bedeutung zu. Grund genug, daß wir uns einmal näher mit interessanten Programmiertechniken rund ums Datum auseinandersetzen.

Dabei orientieren wir uns am besten an folgendem Demoprogramm:

Listing 3.10: DATUM-DEMO

```
100 REM ********************************
110 REM *                              *
120 REM * D E M O   Z U M   D A T U M *
130 REM *                              *
140 REM ********************************
150 REM *                              *
160 REM *  (W) BY FLORIAN MUELLER      *
170 REM *                              *
180 REM *      29. SEPTEMBER 1987      *
190 REM *                              *
200 REM ********************************
210

220 DATA 31,28,31,30,31,30,31,31,30,31,30,31
230 DIM MM%(12),R$(3),WT$(6)
    FOR I=1 TO 12
    READ MM%(I)
    NEXT
    FORI=0TO6
    READWT$(I)
    NEXT
240 DP$="30.09.1987"
    DA$="29.09.1987"
250 P=0
    FORF=1TO3
    R$(F)=""
    NEXT
    R$(0)="(RVS)"
    PRINT"(CLR)"
260 PRINT"(HOME)TAGESDATUM
     "DA$
270 PRINT"(DOWN)PRUEFDATUM
     "DP$
280

290 PRINT"(DOWN)(DOWN)(DOWN)(SWUC)(DISH)"TAB(14)"┌─────────────┐"
300 PRINTTAB(14)"| M E N U E |"
310 PRINTTAB(14)"└─────────────┘"
320 PRINT"(DOWN)(DOWN)"TAB(11)R$(0)"NEUE DATEN EINGEBEN"
330 PRINT"(DOWN)"TAB(11)R$(1)"WOCHENTAG ERMITTELN"
340 PRINT"(DOWN)"TAB(7)R$(2)"PRUEFUNG AUF PLAUSIBILITAET"
350 PRINT"(DOWN)"TAB(8)R$(3)"SPEICHERUNG KOMPRIMIEREN"
360 GETA$
    IFA$=""THEN360
370 IF A$=CHR$(13)THENON P+1 GOTO1000,2000,3000,4000
380 R$(P)=""
390 IF A$="(DOWN)" THEN P=P+1+4*(P=3)
400 IF A$="(UP)" THEN P=P-1-4*(P=0)
410 IF A$="(HOME)" THEN P=0
420 R$(P)="(RVS)"
430 GOTO260
440

450 DATASONNTAG,MONTAG,DIENSTAG,MITTWOCH,DONNERSTAG,FREITAG,SAMSTAG
1000 PRINT"(DOWN)(DOWN)AKTUELLES DATUM IM FORMAT TT.MM.JJJJ
     "

1010 OPEN1,0
     INPUT#1,DA$
     CLOSE1
     PRINT
1020 PRINT"(DOWN)PRUEFDATUM IM SELBEN FORMAT TT.MM.JJJJ
     "

1030 OPEN1,0
     INPUT#1,DP$
     CLOSE1
     PRINT
```

```
1040 GOTO250
1050

2000 PRINT"(DOWN)(DOWN)WOCHENTAG FUER "DP$"
      ";
2010 T=VAL(LEFT$(DP$,2))
     M=VAL(MID$(DP$,4,2))
     J=VAL(RIGHT$(DP$,4))
2020 N=INT(365.25*(J+(M<3)))+INT(30.6*(M+1-(M<3)*12))+T-2
     PRINTWT$(N-INT(N/7)*7)
2030 PRINT"(DOWN)<RETURN> ";
     POKE 204,0
2040 GET A$
     IF A$<>CHR$(13) THEN 2040
2050 POKE 207,0
     POKE 204,1
     GOTO 250
2060

3000 T=VAL(LEFT$(DP$,2))
     M=VAL(MID$(DP$,4,2))
     J=VAL(RIGHT$(DP$,4))
3010 SJ=(J/4-INT(J/4))=0
     REM SJ=-1 BEI SCHALTJAHR, SONST SJ=0
3020 JD=VAL(RIGHT$(DA$,4))
     MD=VAL(MID$(DA$,4,2))
     TD=VAL(LEFT$(DA$,2))
3030 IF J<JD THEN 3500
3040 IF M=2 AND T=29 AND SJ THEN 3400
3050 IF M=2 AND T=29 AND NOT SJ THEN 3500
3060 IF M<1 OR M>12 THEN 3500
3070 IF T<1 OR T>MM%(M) THEN 3500
3080 IF J=JD AND M<MD THEN 3500
3090 IF J=JD AND M=MD AND T<=TD THEN 3500
3400 WF=-1
     GOTO 3550
3500 WF=0
3550 PRINT"(DOWN)DAS EINGEGEBENE DATUM ("DP$") IST(DOWN)"
3600 IF WF THEN PRINT"PLAUSIBEL."
     GOTO 2030
3610 IF NOT WF THEN PRINT"NICHT PLAUSIBEL!"
     GOTO 2030
3620

4000 PRINT"(CLR)NUN WIRD DAS DATUM ("DP$")"
4010 PRINT"(DOWN)IN EINEM STRING VON 3 BYTES LAENGE"
4020 PRINT"(DOWN)GESPEICHERT. BEDINGUNG
      JAHRESZAHL MUSS"
4030 PRINT"(DOWN)ZWISCHEN 1970 UND 1995 LIEGEN."
4040 T=VAL(LEFT$(DP$,2))
     M=VAL(MID$(DP$,4,2))
     J=VAL(RIGHT$(DP$,4))
4050 DK$=CHR$(T+65)+CHR$(M+65)+CHR$(J-1900)
4060 PRINT"(DOWN)(DOWN)DRUECKEN SIE NUN ZWEIMAL <RETURN>."
4070 PRINT"(DOWN)(DOWN)(DOWN)CLR
     DK$="CHR$(34)DK$CHR$(34)
4080 PRINT"(DOWN)(DOWN)GOTO 5000
     ENTSCHLUESSELUNG VON DK$"
4090 END
4100 REM *** NUN BEFINDET SICH DER
4110 REM *** C64 IM DIREKTMODUS.
4120 REM *** LEDIGLICH DIE VARIABLE
4130 REM *** DK$ ERLAUBT DIE SOFORTIGE
4140 REM *** REKONSTRUKTION DES DATUMS
4150 REM *** DP$.
4160
```

```
5000 T=ASC(LEFT$(DK$,1))-65
5010 M=ASC(MID$(DK$,2))-65
5020 J=1900+ASC(RIGHT$(DK$,1))
5030 DP$=RIGHT$(STR$(T),2)+"."+RIGHT$(STR$(M),2)+"."+RIGHT$(STR$(J),4)
5040
5050 PRINT"(DOWN)(RVS)DAS DATUM WAR
      "DP$
5060 END

READY.
```

```
100 REM ********************************
110 REM *                              *
120 REM * D E M O   Z U M   D A T U M *
130 REM *                              *
140 REM ********************************
150 REM *                              *
160 REM *  (W) BY FLORIAN MUELLER      *
170 REM *                              *
180 REM *      29. SEPTEMBER 1987      *
190 REM *                              *
200 REM ********************************
210 :
220 DATA 31,28,31,30,31,30,31,31,30,31,30,31
230 DIM MM%(12),R$(3),WT$(6):FOR I=1 TO 12:READ MM%(I):NEXT:FORI=0TO6:READWT$(I)
:NEXT
240 DP$="30.09.1987":DA$="29.09.1987"
250 P=0:FORF=1TO3:R$(F)="":NEXT:R$(0)="(RVS)":PRINT"(CLR)"
260 PRINT"(HOME)TAGESDATUM: "DA$
270 PRINT"(DOWN)PRUEFDATUM: "DP$
280 :
290 PRINT"(DOWN)(DOWN)(DOWN)(SWUC)(DISH)"TAB(14)"╭───────────╮"
300 PRINTTAB(14)"| M E N U E |"
310 PRINTTAB(14)"╰───────────╯"
320 PRINT"(DOWN)(DOWN)"TAB(11)R$(0)"NEUE DATEN EINGEBEN"
330 PRINT"(DOWN)"TAB(11)R$(1)"WOCHENTAG ERMITTELN"
340 PRINT"(DOWN)"TAB(7)R$(2)"PRUEFUNG AUF PLAUSIBILITAET"
350 PRINT"(DOWN)"TAB(8)R$(3)"SPEICHERUNG KOMPRIMIEREN"
360 GETA$:IFA$=""THEN360
370 IF A$=CHR$(13)THENON P+1 GOTO1000,2000,3000,4000
380 R$(P)=""
390 IF A$="(DOWN)" THEN P=P+1+4*(P=3)
400 IF A$="(UP)" THEN P=P-1-4*(P=0)
410 IF A$="(HOME)" THEN P=0
420 R$(P)="(RVS)"
430 GOTO260
440 :
450 DATASONNTAG,MONTAG,DIENSTAG,MITTWOCH,DONNERSTAG,FREITAG,SAMSTAG
1000 PRINT"(DOWN)(DOWN)AKTUELLES DATUM IM FORMAT TT.MM.JJJJ:"
1010 OPEN1,0:INPUT#1,DA$:CLOSE1:PRINT
1020 PRINT"(DOWN)PRUEFDATUM IM SELBEN FORMAT TT.MM.JJJJ:"
1030 OPEN1,0:INPUT#1,DP$:CLOSE1:PRINT
1040 GOTO250
1050 :
2000 PRINT"(DOWN)(DOWN)WOCHENTAG FUER "DP$": ";
2010 T=VAL(LEFT$(DP$,2)):M=VAL(MID$(DP$,4,2)):J=VAL(RIGHT$(DP$,4))
2020 N=INT(365.25*(J+(M<3)))+INT(30.6*(M+1-(M<3)*12))+T-2:PRINTWT$(N-INT(N/7)*7)
2030 PRINT"(DOWN)<RETURN> ";:POKE 204,0
2040 GET A$:IF A$<>CHR$(13) THEN 2040
2050 POKE 207,0:POKE 204,1:GOTO 250
2060 :
```

```
3000 T=VAL(LEFT$(DP$,2)):M=VAL(MID$(DP$,4,2)):J=VAL(RIGHT$(DP$,4))
3010 SJ=(J/4-INT(J/4))=0:REM SJ=-1 BEI SCHALTJAHR, SONST SJ=0
3020 JD=VAL(RIGHT$(DA$,4)):MD=VAL(MID$(DA$,4,2)):TD=VAL(LEFT$(DA$,2))
3030 IF J<JD THEN 3500
3040 IF M=2 AND T=29 AND SJ THEN 3400
3050 IF M=2 AND T=29 AND NOT SJ THEN 3500
3060 IF M<1 OR M>12 THEN 3500
3070 IF T<1 OR T>MM%(M) THEN 3500
3080 IF J=JD AND M<MD THEN 3500
3090 IF J=JD AND M=MD AND T<=TD THEN 3500
3400 WF=-1:GOTO 3550
3500 WF=0
3550 PRINT"(DOWN)DAS EINGEGEBENE DATUM ("DP$") IST(DOWN)"
3600 IF WF THEN PRINT"PLAUSIBEL.":GOTO 2030
3610 IF NOT WF THEN PRINT"NICHT PLAUSIBEL!":GOTO 2030
3620 :
4000 PRINT"(CLR)NUN WIRD DAS DATUM ("DP$")"
4010 PRINT"(DOWN)IN EINEM STRING VON 3 BYTES LAENGE"
4020 PRINT"(DOWN)GESPEICHERT. BEDINGUNG: JAHRESZAHL MUSS"
4030 PRINT"(DOWN)ZWISCHEN 1970 UND 1995 LIEGEN."
4040 T=VAL(LEFT$(DP$,2)):M=VAL(MID$(DP$,4,2)):J=VAL(RIGHT$(DP$,4))
4050 DK$=CHR$(T+65)+CHR$(M+65)+CHR$(J-1900)
4060 PRINT"(DOWN)(DOWN)DRUECKEN SIE NUN ZWEIMAL <RETURN>."
4070 PRINT"(DOWN)(DOWN)(DOWN)CLR:DK$="CHR$(34)DK$CHR$(34)
4080 PRINT"(DOWN)(DOWN)GOTO 5000:ENTSCHLUESSELUNG VON DK$(UP)(UP)(UP)(UP)(UP)(UP
)"
4090 END
4100 REM *** NUN BEFINDET SICH DER
4110 REM *** C64 IM DIREKTMODUS.
4120 REM *** LEDIGLICH DIE VARIABLE
4130 REM *** DK$ ERLAUBT DIE SOFORTIGE
4140 REM *** REKONSTRUKTION DES DATUMS
4150 REM *** DP$.
4160 :
5000 T=ASC(LEFT$(DK$,1))-65
5010 M=ASC(MID$(DK$,2))-65
5020 J=1900+ASC(RIGHT$(DK$,1))
5030 DP$=RIGHT$(STR$(T),2)+"."+RIGHT$(STR$(M),2)+"."+RIGHT$(STR$(J),4)
5040 :
5050 PRINT"(DOWN)(RVS)DAS DATUM WAR: "DP$
5060 END
```

Hierbei wird neben dem sogenannten »aktuellen Datum« in DA$, also einem Anhaltspunkt für Datumsvergleiche, jeweils ein »Prüfdatum« in DP$ verwendet. Das Hauptmenü, in welchem Sie mit den Cursortasten das Inversfeld bewegen und durch <RETURN> die aktuell hervorgehobene Funktion aufrufen, bietet folgende Demos an:

- **Neue Daten eingeben (Zeilen 1000–1050)**

Dies ist Ausgangspunkt für alle weiteren Operationen mit diesen Daten. Voreingestellt sind der 30.09.1987 als Prüfdatum und der 29.09.1987 als aktuelles Datum. Die Eingabe erfolgt jeweils im Format TT.MM.JJJJ, unbenutzte Stellen werden also durch Nullen aufgefüllt. Diese Eingaberoutine prüft zunächst nicht, ob die aktuellen Eingaben im Bereich des Sinnvollen liegen.

- **Wochentag ermitteln (Zeilen 2000–2060)**

Hierbei erscheint für das aktuelle Prüfdatum DP$ der jeweilige Wochentag; Berechnungsgrundlage ist Einzeiler #35; anstelle der nichtssagenden Kennziffer für den jeweils ermittelten Wochentag erscheint hierbei der Klartext, welcher für alle Wochentage im Array WT$() steht. Interessant ist auch, daß beim Warten auf <RETURN> der Cursor kurzfristig eingeschaltet wird (Zeile 2030, s. 2.5); in 2050 wird er vor Rücksprung ins Hauptmenü wieder ausgeschaltet, weil sonst reverse Leerzeichen wie »Flecken« am Bildschirm haften bleiben.

- **Prüfung auf Plausibilität (Zeilen 3000–3620)**

Der Plausibilitätstest identifiziert nicht nur Eingabefehler und Daten, die nach logischen Gesetzen unmöglich sind (31. Februar usw.), sondern stellt auch sicher, daß das eingegebene Prüfdatum DP$ erst nach (!) dem aktuellen Datum DA$ liegt. Die Plausibilitätsprüfung reicht für die Praxis völlig aus; für wissenschaftliche Zwecke wäre eine Ergänzung sicherlich notwendig, um die Reformen des Gregorianischen Kalenders oder die Situation von ausfallenden Schaltjahren (welche in den nächsten 2000 Jahren ohnehin nicht eintritt) auch noch zu berücksichtigen. Es wäre jedoch absolut sinnlos, ein normales C64-Programm mit solchen für den vernünftigen Anwender nutzlosen Mätzchen auszustatten.

- **Speicherung komprimieren (Zeilen 4000–4160 sowie 5000–5060)**

Hier lernen Sie, wie das Datum in einen String DK$ von genau drei Zeichen Länge gerettet wird. Dieser String enthält herkömmliche ASCII-Zeichen, zu deren Eingabe Sie nicht einmal den Quote Mode heranziehen müssen. Verwendbar sind hier jedoch nur Daten zwischen 1970 und 1995; Sie müssen also nach Ablauf dieser Zeit ein neues Programm zu diesem Thema entwickeln, sofern Sie noch mit dem C64 arbeiten wollen...

3.2.4 Zahlenformate umrechnen

Für bestimmte Anwendungen ist die Unterstützung verschiedener Zahlensysteme unvermeidbar. Deshalb wurden in 2.1 als Einzeiler #27 und #28 zwei »Routinen« vorgestellt, welche dieses Problem für alle Zahlensysteme (zumindest, solange höhere Ziffern als 10 durch Buchstaben dargestellt werden) lösen. Der größte Vorteil dieser Einzeiler ist gleichzeitig ein gravierender Nachteil: Ausgehend von der frei wählbaren Basis B ist die Routine etwas umständlicher programmiert, als sie es für eine konstante Basis wäre. Bei allen Zahlensystemen nach dem Dezimalsystem ist hierbei einfach die Basis B durch den entsprechenden Zahlenwert zu ersetzen:

```
10 Z$="":FORP=0TO0:D=D/16:S=(D-INT(D))*16:Z$=CHR$(55+S+7*(S<10))+Z$:P
   =-D:NEXT
```

sowie

```
10 D=0:FORS=1TOLEN(Z$):H=ASC(MID$(Z$,S))-48:D=D*16+H+7*(H>9):NEXT
```

sind die entsprechenden Einzeiler #27 und #28, speziell für das Hexadezimalsystem (B wurde durch 16 ersetzt). Der Vorteil besteht in einer größeren Übersichtlichkeit des Listings sowie der erleichterten Verwendung, weil die Basis nicht mehr als Parameter mitzuteilen ist.
Erhebliche Vorteile ergeben sich jedoch erst bei Berücksichtigung von Zahlensystemen mit einer kleineren Basis als 10. Folgende Einzeiler sind speziell auf das Binärsystem zugeschnitten, und gestalten sich sehr einfach, weil die Berücksichtigung von Buchstaben als Ziffern für das Zweiersystem entfällt:

```
10 Z$="":FORP=0TO0:D=D/2:S=2*(D-INT(D)):Z$=CHR$(48+S)+Z$:P=-D:NEXT
```

sowie

```
10 D=0:FORS=1TOLEN(Z$):H=ASC(MID$(Z$,S))-48:D=2*D+H:NEXT
```

Damit gestaltet sich doch die Verständigung zwischen den einzelnen Zahlensystemen sehr bequem, oder?

3.2.5 INSTR$-Funktion

Bei der Arbeit mit Strings ist es oft von größter Wichtigkeit, darüber informiert zu sein, ob sich ein Zeichen oder eine bestimmte Zeichenkette in einem anderen String befindet. In anderen Basic-Dialekten trägt eine dafür vorgesehene Funktion den Namen INSTR$. Dennoch brauchen wir in Basic 2.0 nicht zu verzweifeln, denn folgendes Programm enthält in den Zeilen 400–440 eine praktikable Ersatzlösung:

Listing 3.11: INSTR-DEMO

```
100 REM ******************************
110 REM *                            *
120 REM *   ERSATZROUTINE IN BASIC   *
130 REM *                            *
140 REM * FUER DIE FEHLENDE FUNKTION *
150 REM *                            *
160 REM *       I N S T R $          *
170 REM *                            *
180 REM * (GIBT AUSKUNFT, OB - UND,  *
190 REM *                            *
200 REM *  SOFERN JA: WO - EIN       *
210 REM *                            *
220 REM *  STRING IN EINEM ANDEREN   *
230 REM *                            *
240 REM *  ENTHALTEN IST/SEIN KANN)  *
250 REM *                            *
260 REM ******************************
270 :
280 :
300 GOTO500:REM ZUM DEMONSTRATIONSTEIL SPRINGEN
310 :
350 REM *** ZEILEN 400-440: ***
360 REM *** SUCHE VON A$    ***
370 REM *** IM STRING AA$   ***
380 :
400 IF LEN(AA$)<LEN(A$)THENI=0:RETURN
410 FORI=1TOLEN(AA$)-LEN(A$)+1
```

```
420 IFMID$(AA$,I,LEN(A$))<>A$THENNEXT
430 IF I>LEN(AA$)-LEN(A$)+1 THEN I=0
440 RETURN
450 :
460 :
500 INPUT "HAUPTSTRING";AA$
510 INPUT "TEILSTRING";A$
520 GOSUB 400
530 IF I THEN 600
540 PRINT"(DOWN)DER TEILSTRING "CHR$(34)A$CHR$(34)
550 PRINT"(DOWN)IST IM HAUPTSTRING "CHR$(34)AA$CHR$(34)
560 PRINT"(DOWN)NICHT ENTHALTEN.(DOWN)(DOWN)(DOWN)"
570 GOTO500
600 PRINT"(DOWN)DER TEILSTRING "CHR$(34)A$CHR$(34)
610 PRINT"(DOWN)IST IM HAUPTSTRING "CHR$(34)AA$CHR$(34)
620 PRINT"(DOWN)AB DER POSITION"I"ENTHALTEN.(DOWN)(DOWN)(DOWN)"
630 GOTO500
```

Ab Zeile 500 steht ein selbsterklärender Demonstrationsteil. Daran erkennen Sie leicht die Übergabeparameter der INSTR$-Routine:

- A$ ist der Teilstring, welcher in
- AA$ gesucht wird

Zurückgegeben wird die Position in I, wobei 0 gleichbedeutend mit »Teilstring nicht in Hauptstring gefunden« ist.

3.2.6 Teilstringzuweisung

In Basic 2.0 ist MID$ lediglich eine Funktion, nicht jedoch ein Zuweisungsbefehl: MID$(A$,I,1)=»X« ergibt einen SYNTAX ERROR. Bereits das Basic 7.0 des C 128 läßt jedoch zu, MID$ links vom Zuweisungszeichen »=« zu plazieren und dadurch nur einen Teil eines Strings zu verändern.
Bei Basic 2.0 erfordert es einen gewissen Einfallsreichtum, zum selben Ergebnis zu kommen. Dabei ist nämlich der jeweils zu ändernde String A$ in einen linken und rechten Reststring aufzuteilen. Nehmen wir folgendes Beispiel: A$ enthalte den String »MAIER«, und wir wollen das zweite und dritte Zeichen in A$ durch den String N$=»EY« ersetzen, wonach A$=»MEYER« sein muß; P enthält 2, die Einfügeposition für N$ in A$. Teilen wir also A$ in A1$ und A2$ auf:

```
A1$=LEFT$(A$,P-1):A2$=MID$(A$,P+LEN(N$))
```

Anschließend ist A1$=»M«, A2$=»ER«.

Nun bleibt lediglich, das Ergebnis zu einem einzigen String zu verknüpfen, welcher wieder A$ heißen soll und dadurch den alten Inhalt von A$ automatisch überschreibt:

```
        A$=A1$+N$+A2$
also:   A$="M"+"EY"+"ER", A$="MEYER"
```

Wesentlich einfacher ist das Ersetzen eines linken oder rechten Teilstrings, da hierbei nur jeweils der Reststring aus der entgegengesetzten Richtung zu definieren ist, bevor dieser Rest mit dem Einfügestring verknüpft wird. Das angeführte Grundgerüst für MID$-Zuweisungen dürfte jedoch am häufigsten vorkommen.

4 Maschinenroutinen für Basic oder: Zwei Welten ergänzen sich

Beschränkt man sich bei der Programmierung auf die Programmiersprache Basic, stößt man oft an deren Leistungsgrenzen, die eben in manchen Bereichen (Grafik!) niedriggesteckt sind. Die Maschinenprogrammierung ist jedoch verständlicherweise nicht jedermanns Sache, ganz abgesehen vom Mehraufwand bei der Programmentwicklung. Deshalb sind für jeden Basic-Programmierer fertige Maschinenroutinen zur direkten Einbindung in Basic-Rahmenprogramme oftmals eine Offenbarung und vermitteln weitere Faszination am C64. In 2.4 haben Sie bereits bewundern können, welche interessanten Effekte durch Maschinenroutinen in Basic bewirkt werden.
Dennoch ist selbst die Anwendung von fertigen Maschinenprogrammen für den Basic-Freak keine ganz einfache Angelegenheit und führt bei intensivem Gebrauch zu erheblichen Problemen. Deshalb finden Sie in diesem Kapitel zwar nur einen Bruchteil aller Maschinenroutinen, die ich Ihnen anbieten könnte; dafür aber handelt es sich meines Erachtens um die sinnvollsten Routinen zu diesem Zweck.

Nun ein ganz wichtiger Hinweis:

Die meisten Programme sind nur als einzige Maschinenroutine innerhalb eines Basic-Programms lauffähig; Überschneidungen führen zu Abstürzen und sind unbedingt zu vermeiden!
Dies ist eine Einschränkung, die durch die Speicherverwaltung des C64 entsteht und allen Maschinenroutinen gemeinsam ist. Nicht zuletzt deshalb sind Basic-Module immer »handlicher« als Maschinenprogramme.
Die zahlreichen Programmbeschreibungen sollen Maschinensprache-Freaks jedoch helfen, sich eigener Anpassungen zu bedienen.

4.1 Verlangsamte Bildschirmausgabe

Durch POKEn von entsprechenden Werten in die Adresse 56325 läßt sich die Ablaufgeschwindigkeit des C64 durch Veränderung der IRQ-Anzahl in einem bestimmten Intervall (siehe Kapitel 10) regulieren; damit verlangsamt bzw. beschleunigt man jedoch jeweils den gesamten Programmablauf.

Für viele Zwecke bietet es sich nun an, nur aus der Zeichenausgabe (PRINT usw.) das Tempo zu nehmen:

Listing 4.1 a: Basic-Lader für BILDSCH.LANGSAM

```
10 PRINT"<CLR>":FOR I=0 TO 43:READ X:POKE 1024+I,X:NEXT:SYS 1024
11 DATA 160,0,185,24,4,153,236,207,200,192,20,208,245,160,236,140,38,3,160
12 DATA 207,140,39,3,96
13 DATA 72,152,72,160,100,169,255,233,1,208,252,136,208,247,104,1668,104,76
14 DATA 202,241
```

Dabei ist folgendes zu beachten:

- Dieses Maschinenprogramm läßt sich ohne Probleme in eigene Programme einbinden; dazu sind lediglich die Zeilennummern anzupassen. Achten Sie jedoch auf die richtige Reihenfolge der DATA-Zeilen!
- Zunächst schreibt Zeile 10 das Maschinenprogramm in den C64-Bildschirmspeicher, weshalb nach seiner Initialisierung der Bildschirm gelöscht werden muß. Das Programm verschiebt sich dann selbst an Adresse 53228.
- Alle Ausgabevektoren werden auf die eigene Routine gerichtet, welche im wesentlichen nur eine Verzögerungsschleife beinhaltet.
- Durch <RUN/STOP RESTORE> oder POKE806,202:POKE807,241 schaltet man die Routine aus.
- POKE53232,x reguliert die Ausgabegeschwindigkeit, wobei 1 den Normalwert darstellt.

Für Maschinenprogrammierer ist sicher das Listing der Routine von Interesse:

Listing 4.1 b: Maschinenroutine zu BILDSCH.LANGSAM

```
; Einsprung: SYS 1024 ($0400)

. 0400  A0 00     LDY #$00          ; Kopierschleife:
. 0402  B9 18 04  LDA $0418,Y       ;
. 0405  99 EC CF  STA $CFEC,Y       ; überträgt
. 0408  C8        INY               ; Routine von
. 0409  C0 14     CPY #$14          ; $0418 bis $042b
. 040B  D0 F5     BNE $0402         ; nach $cfec

. 040D  A0 EC     LDY #$EC   <($CFEC)  ; stellt
. 040F  8C 26 03  STY $0326            ; Vektor IBSOUT
. 0412  A0 CF     LDY #$CF   >($CFEC)  ; (für BSOUT $ffd2)
. 0414  8C 27 03  STY $0327            ; auf neue Routine
. 0417  60        RTS                  ; um
------------------------------
```

```
; die nun folgende Routine wird bei $0400-$040b nach $cfec verschoben
; dahin weist dann der Vektor IBSOUT

. 0418  48        PHA          ; Akku und
. 0419  98        TYA          ; Y-Register
. 041A  48        PHA          ; auf Stapel retten

. 041B  A0 64     LDY #$64     ; Zähler für äußere Schleife
. 041D  A9 FF     LDA #$FF     ; Zähler für innere Schleife
. 041F  E9 01     SBC #$01     ; innere Schleife weiterzählen
. 0421  D0 FC     BNE $041F    ; noch nicht fertig (Z=0): weiter
. 0423  88        DEY          ; äußere Schleife weiterzählen
. 0424  D0 F7     BNE $041D    ; noch nicht fertig (Z=0): weiter
. 0426  68        PLA          ; Akku und
. 0427  A8        TAY          ; Y-Register
. 0428  68        PLA          ; wiederherstellen

. 0429  4C CA F1 JMP $F1CA    ; alte BSOUT-Routine auslösen
--------------------------------
```

4.2 RE-CLR, das OLD für Variablen

Wie Sand am Meer gibt es OLD-Routinen, also Programme zur Wiederherstellung gelöschter, aber noch im Speicher befindliche Basic-Programme (siehe Einzeiler #2 und Kapitel 5). Das Programm »RE-CLR« ist jedoch auch in der Lage, sämtliche Variablen, Strings und Felder zu restaurieren. Es wird mit

```
LOAD "RE-CLR",8,1
```

geladen, woraufhin man normalerweise NEW eingeben sollte. Bei Bedarf startet

```
SYS 49152
```

die RE-CLR-Routine; es erscheint zwar ein OUT OF MEMORY ERROR, doch dieser kann getrost ignoriert werden (er entsteht nur wegen Umstellen der Basic-Zeiger). Diese Routine belegt 115 Byte Speicherplatz, ist also phänomenal kurz.
Es können, wie gesagt, alle Werte und sogar definierte Funktionen und Felder zurückgeholt werden. Allerdings bestehen dafür noch drei kleine Bedingungen:

1. Es muß vorher bereits ein Basic-Programm im Speicher gestanden haben (zumindest eine einzige Programmzeile).
2. Es darf seit Löschen der Variablen und des Programms, aber vor Aufrufen der Routine keine Variable eingegeben und kein Feld abgefragt worden sein.
3. Die erste Dimensionierung eines Feldes muß ein Stringfeld sein. Beispiele:
   - DIM E(20),E$(20) muß umgestellt werden zu DIM E$(20),E(20)
   - DIM E(34) muß ergänzt werden zu DIM A$(0),E(34); dabei genügt selbst A$(0) schon zum einwandfreien Funktionieren von RE-CLR. Diese Ergänzung benötigt der Computer nämlich, um zu erkennen, daß in die Array-Behandlung gesprungen werden muß; ansonsten entsteht ein Fehler.

Zur Funktionsweise des Programms:

Der C64 untersucht das erste Byte nach Ende des Basic-Programms; ist dieses ein Buchstabe, prüft er es auf Stringvariable. Andernfalls wird der Zähler einfach um 7 Byte (siehe »C64 für Insider«) erhöht und dieser Eintrag übergangen. Wurde hingegen eine Stringvariable identifiziert, untersucht er Byte 6 und 7. Diese sind bei einem String nämlich immer Null, weil sie hier nicht benötigt werden (siehe »C64 für Insider«); bei einem Feldkopf hingegen ist die Anzahl der Elemente der Dimensionierung des nächsten Feldes darin abgelegt:
Wenn RE-CLR also eine numerische als erste Dimensionierung findet, hält er sie für eine normale numerische Variable – denn eine andere Erkennung ist nach NEW oder CLR nicht mehr zu treffen – und nicht, wie es richtig wäre, für den ersten Feldkopf der Arrays; gibt einen falschen, weil unsinnigen Wert für die meist nicht vorhandene einfache Variable aus und findet auch die restlichen Felder nicht mehr!
Sobald keine Buchstaben mehr zu finden sind, setzt die Garbage Collection (siehe »C64 für Insider«) ein, welche die Stringzeiger wieder korrigiert.
Beachten Sie bitte, daß nach Wiederherstellung folgenden Programms automatisch A$(30), aber auch B$(10) zurückgeholt wird:

```
10 DIM A$(30),B$(10)
20 CLR
30 DIM A$(30)
40 SYS 49152
```

Auch hier ist eine weitere Unterscheidung unmöglich, weil RE-CLR nicht den logischen Ablauf eines Programms verfolgen kann.
Zuletzt noch zur Anpassung des Programms an andere Speicherbereiche: Lediglich die Sprungbefehle

```
C04B JMP C018
C06A JMP C04E
```

sind jeweils anzupassen; ansonsten ist das Programm verschiebbar.

Listing 4.2 ist ein dokumentiertes Assembler-Listing zu RE-CLR:

LABEL	LOC	CODE			STATEMENT		
STPRG	0000				*=	$C000	
Old	C000	A0	01		LDY	#01	
	C002	98			TYA		
	C003	91	2B		STA	(2B),Y	eine »NEW-Null« löschen (mit 1 überschreiben)
	C005	20	33	A5	JSR	A533	Basic-Zeilen neu binden
	C008	18			CLC		
	C009	A5	22		LDA	22	
	C00B	69	02		ADC	#02	
	C00D	85	2D		STA	2D	
	C00F	A5	23		LDA	23	Programmende (= Zeiger +2 Nullen) überlesen
	C011	69	00		ADC	#00	
	C013	85	2E		STA	2E	
	C015	20	60	A6	JSR	A660	zum CLR-Befehl
Variablen	C018	A0	00		LDY	#00	
	C01A	B1	2F		LDA	(2F),Y	1. Byte nach Programmende
	C01C	AA			TAX		retten
	C01D	29	7F		AND	#7F	Bit 7 löschen
	C01F	20	13	B1	JSR	B113	Buchstabe?
	C022	90	49		BCC	C06D	nein, denn fertig
	C024	8A			TXA		
	C025	0A			ASL		war Bit 7 gesetzt?
	C026	B0	10		BCS	C038	ja, kein String (sondern Integervariable oder Funktion),
	C028	C8			INY		Arrayerkennungsschleife überspringen
	C029	B1	2F		LDA	(2F),Y	2. Namensbyte
	C02B	10	0B		BPL	C038	kleiner als $80, also REAL-Variable, kein String
	C02D	A0	05		LDY	#05	
	C02F	B1	2F		LDA	(2F),Y	6. Byte des Strings
	C031	D0	1B		BNE	C04E	ungleich Null => Array-Feldkopf, kein String
	C033	C8			INY		
	C034	B1	2F		LDA	(2F),Y	7. Byte des Strings
	C036	D0	16		BNE	C04E	ungleich Null => Array-Feldkopf, kein String
	C038	A0	07		LDY	#07	
	C03A	E6	2F		INC	2F	
	C03C	D0	02		BNE	C040	
	C03E	E6	30		INC	30	7 Bytes überlesen
	C040	88			DEY		
	C041	D0	F7		BNE	C03A	
	C043	A5	2F		LDA	2F	
	C045	85	31		STA	31	
	C047	A5	30		LDA	30	Array-Ende = Variablen-Ende
	C049	85	32		STA	32	
	C04B	4C	18	C0	JMP	C018	und weitermachen; nächstes Element prüfen
ARRAY	C04E	A0	00		LDY	#00	
	C050	B1	31		LDA	(31),Y	1. Byte nach Variablenende
	C052	29	7F		AND	#7F	Bit 7 löschen
	C054	20	13	B1	JSR	B113	Buchstabe?
	C057	90	14		BCC	C06D	nein, dann fertig: keine Felder mehr
	C059	C8			INY		
	C05A	C8			INY		
	C05B	18			CLC		
	C05C	B1	31		LDA	(31),Y	Feldlänge Low-Byte
	C05E	65	31		ADC	31	zu Array-Ende-Zeiger addieren und merken
	C060	AA			TAX		
	C061	C8			INY		
	C062	B1	31		LDA	(31),Y	Feldlänge High-Byte
	C064	65	32		ADC	32	zu Array-Zeiger addieren
	C066	85	32		STA	32	Feld überlesen und
	C068	86	31		STX	31	Array-Zeiger auf Feld-Ende positionieren
	C06A	4C	4E	C0	JMP	C04E	und weitermachen
Ende	C06D	20	26	B5	JSR	B526	Garbage-Collection: Stringzeiger nachstellen
	C070	4C	AE	A7	JMP	A7AE	zurück zur Interpreterschleife

4.3 Zeilenmarker, das Zeilenlineal

Für viele Anwendungsfälle ist es eine nicht zu unterschätzende Hilfe, wenn die aktuelle Zeile – die Zeile, in welcher der Cursor steht – am Bildschirm optisch hervorgehoben wird. Folgendes Programm ist dazu in der Lage:

Listing 4.3a: ZEILENMARKER, Basic-Teil

```
100 REM   ZEILENMARKER                       <021>
110 REM       BY                             <083>
120 REM    ALEXANDER                         <095>
130 REM      PIEL                            <145>
140 REM                                      <202>
150 REM    (C) 1986                          <250>
160 FOR I=49156 TO 49266:READ A:POKE I,A     <072>
170 S=S+A:NEXT                               <005>
180 IF S=12648 THEN 510                      <120>
190 PRINT"DATA-FEHLER":END                   <178>
200 DATA 120,169, 35,141, 20,  3,169         <159>
210 DATA 192,141, 21,  3,173, 17,208         <191>
220 DATA  41,127,141, 17,208,169,  0         <169>
230 DATA 141, 18,208,169,129,141, 26         <055>
240 DATA 208, 88, 96,173, 25,208,141         <162>
250 DATA  25,208, 48,  7,173, 13,220         <228>
260 DATA  88, 76, 49,234,165,214,197         <007>
270 DATA   2,240, 21,133,  2,165,214         <027>
280 DATA 141,  0,192, 24, 42, 42, 42         <034>
290 DATA 105, 49,141,  0,192,105,  9         <186>
300 DATA 141,  1,192,173, 18,208,205         <164>
310 DATA   1,192,176, 15,173,  2,192         <018>
320 DATA 141, 33,208,173,  1,192,141         <146>
330 DATA  18,208, 76,112,192,173,  3         <220>
340 DATA 192,141, 33,208,173,  0,192         <246>
350 DATA 141, 18,208, 76,188,254             <106>
400 REM  EINSCHALTEN MIT                     <209>
410 REM  SYS 49156                           <143>
420 REM  FARBE DES BALKENS :                 <192>
430 REM  POKE 49154,[ FARBCODE ]             <070>
440 REM                                      <248>
450 REM  HINTERGRUNDFARBE NICHT MEHR         <231>
460 REM  POKE 53281,[ FARBCODE ]             <212>
470 REM  SONDERN                             <014>
480 REM  POKE 49155,[ FARBCODE ]             <128>
490 REM                                      <042>
500 REM         DEMO                         <038>
510 POKE 49154,1:POKE 49155,PEEK(53281)      <094>
520 SYS 49156                                <072>
```

Die REM-Zeilen geben bereits Auskunft über die benötigten Anweisungen, welche auch in den Zeilen 500/510 demonstriert werden:

- SYS 49156 schaltet den Zeilenmarker an
- POKE 49154, <Farbe> setzt die Farbe des Balkens

- POKE 49155, <Farbe> stellt fortan die Hintergrundfarbe ein (nicht mehr POKE 53281, <Farbe>)
- <RUN/STOP RESTORE> beendet die Zeilenmarker-Tätigkeit

Das Funktionsprinzip ist einfach und doch effektiv: In einem Raster-Interrupt wird jeweils der 8-Rasterzeilen-Block für die aktuelle Cursorzeile ausgesondert und bekommt einen eigenen Farbwert für den Hintergrund zugewiesen. Jeweils zu Beginn und Ende der Markierung wird ein Raster-IRQ ausgelöst, welcher die richtige Farbe einstellt und das VIC-Register #18 so einstellt, daß bei der nächsten Grenze (Anfang oder Ende) der Markierung wieder ein Raster-IRQ erfolgt.

Listing 4.3b: ZEILENMARKER, Maschinenroutine

```
; Aktivierung der neuen IRQ-Routine:

. C004  78        SEI           ; Interrupt verhindern

. C005  A9 23     LDA #$23   <($C023) ; IRQ-Vektor
. C007  8D 14 03  STA $0314          ; auf Raster-
. C00A  A9 C0     LDA #$C0   <($C023) ; IRQ-Routine
. C00C  8D 15 03  STA $0315          ; umlenken

. C00F  AD 11 D0  LDA $D011          ; VIC-Register
. C012  29 7F     AND #$7F %01111111 ; auf Raster-IRQ
. C014  8D 11 D0  STA $D011          ; einrichten

. C017  A9 00     LDA #$00           ; Rasterzeile auf
. C019  8D 12 D0  STA $D012          ; 0 setzen
. C01C  A9 81     LDA #$81 %10000001 ; Interrupt Control
. C01E  8D 1A D0  STA $D01A          ; Register aktivieren

. C021  58        CLI           ; Interrupt zulassen
. C022  60        RTS           ; Rücksprung ins Basic
------------------------------

. C023  AD 19 D0  LDA $D019     ; Bit für ausgelösten
. C026  8D 19 D0  STA $D019     ; Raster-IRQ holen
. C029  30 07     BMI $C032     ; gesetzt (N=1): Markierung!

. C02B  AD 0D DC  LDA $DC0D     ; Kontrollregister auslesen
. C02E  58        CLI           ; IRQ wieder zulassen
. C02F  4C 31 EA  JMP $EA31     ; alte IRQ-Routine anspringen
------------------------------

; Markierung im IRQ ziehen

. C032  A5 D6     LDA $D6       ; Nummer der Cursorzeile holen
. C034  C5 02     CMP $02       ; mit letztem Wert vergleichen
```

```
. C036  F0 15     BEQ $C04D      ; gleich (Z=1): nicht neuberechnen
. C038  85 02     STA $02        ; als letzten Wert merken
. C03A  A5 D6     LDA $D6        ; Nummer der Cursorzeile holen
. C03C  8D 00 C0  STA $C000      ; und in $c000 sichern
. C03F  18        CLC            ; Carry löschen
. C040  2A        ROL            ; Akku (Nr. der Cursorzeile)
. C041  2A        ROL            ; mit 8 (Rasterzeilen/Textzeile)
. C042  2A        ROL            ; multiplizieren
. C043  69 31     ADC #$31       ; oberen Bildschirmrand als
                                 ; #49 (Rasterzeile) addieren

. C045  8D 00 C0  STA $C000      ; Ergebnis als Rasterzeile merken
. C048  69 09     ADC #$09       ; End-Rasterzeile für Markierung
. C04A  8D 01 C0  STA $C001      ; berechnen und in $C001 merken

. C04D  AD 12 D0  LDA $D012      ; Nummer der Rasterzeile holen
. C050  CD 01 C0  CMP $C001      ; und mit Endzeile vergleichen
. C053  B0 0F     BCS $C064      ; überschritten (C=1): wieder den
                                 ; normalen Hintergrund einstellen

; Markierung einschalten (ab Anfangs-Rasterzeile)

. C055  AD 02 C0  LDA $C002      ; Farbe der Markierung holen
. C058  8D 21 D0  STA $D021      ; und als Hintergrund setzen
. C05B  AD 01 C0  LDA $C001      ; End-Rasterzeile laden
. C05E  8D 12 D0  STA $D012      ; und als Rasterzeile für
                                 ; nächsten IRQ vormerken
. C061  4C 70 C0  JMP $C070      ; Raster-IRQ beenden
------------------------------

; Markierung ausschalten (nach End-Rasterzeile)

. C064  AD 03 C0  LDA $C003      ; Normalfarbe holen
. C067  8D 21 D0  STA $D021      ; und als Hintergrund setzen
. C06A  AD 00 C0  LDA $C000      ; Anfangs-Rasterzeile laden
. C06D  8D 12 D0  STA $D012      ; und als Rasterzeile für
                                   nächsten IRQ vormerken

. C070  4C BC FE  JMP $FEBC      ; Raster-IRQ beenden
------------------------------
```

4.4 Komprimierte Datenspeicherung

Schreiben Sie öfters Programme, bei denen Sie größere Datenmengen verwalten müssen? Mit dem »Daten-Packer« wird der Aufwand an Ladezeit und Speicherplatz auf Diskette um 25 Prozent reduziert.
Das Programm »PACKROUTOBJ« kürzt Strings auf 3/4 der normalen Länge. Wenn man zum Beispiel einen String mit 12 Zeichen Inhalt hat und den »Daten-Packer« darauf losläßt, ist der String bei unverändertem Inhalt nur noch neun Zeichen lang.
Damit man ihn jedoch ansehen bzw. sonstig verarbeiten kann, muß er natürlich erst wieder in lesbare Form gebracht, also »entpackt« werden.
Bei selbstgeschriebenen Adreßverwaltungen läßt sich das gut anwenden: Bevor man die meist in Strings enthaltenen Adressen auf Diskette ablegt, »packt« man sie und dann erst werden sie gespeichert. So spart man eine beachtliche Menge wertvollen Diskettenplatz und Ladezeit.
Nach dem Einladen müssen die Adressen dann wieder »entpackt« werden, bevor man mit ihnen weiterarbeiten kann.
Das Programm wird mit LOAD"PACKROUTOBJ",8,1 von Diskette geladen, worauf man NEW eingeben sollte, und ist sofort einsatzbereit:

```
SYS 49152,A$,B$
```

dient zum »Packen« des Strings A$, woraufhin B$ die gepackte Version enthält. Achtung: Sie dürfen nur Variablen, nicht konstante Zeichenketten verwenden!

```
SYS 49339,C$,D$
```

bewirkt das Gegenteil: In C$ steht zunächst der »gepackte« String, in D$ ist anschließend der »entpackte« String zur Weiterverarbeitung bereit. Anstelle von A$, B$, C$ und D$ dürfen Sie natürlich jeden anderen Variablennamen einsetzen.

Zur Funktionsweise:

Das geht doch nicht mit rechten Dingen zu, wird der eine oder andere Leser jetzt vielleicht denken. Nun, dieses Programm kürzt beispielsweise 4 Byte auf 3 Byte. Ein Byte besteht bekanntlich aus 8 Bit, bei Datenstrings werden jedoch nur 6 Bit zur Darstellung der ASCII-Werte benötigt. Mit diesen 6 Bit kann man alle alphanumerischen Zeichen (Buchstaben und Zahlen) sowie die wichtigsten Sonderzeichen darstellen.
Der Algorithmus des Programms besteht nun darin, vom ASCII-Wert des zu packenden Bytes 32 abzuziehen, um eine Darstellung mit 6 Bit zu erreichen. Dann wird jedes vierte Byte auf die vorhergehenden 3 Byte verteilt, bei denen die letzten 2 Bit nach unserer Subtraktion nicht mehr nötig sind.
»Überhang-Bytes«, also die letzten Zeichen eines Strings, die keine vollen Bytes mehr ergeben, werden nicht gepackt, sondern einfach unverändert an den String angehängt. Beim Entpacken läuft der Vorgang in umgekehrter Richtung.

Merken sollten Sie sich auf jeden Fall: Wenn der Daten-Packer nicht nach Wunsch arbeitet, so haben Sie in Ihren Strings unerlaubte Zeichen (Steuerzeichen oder Grafiksymbole) verwendet. Dies funktioniert deshalb nicht, weil der Daten-Packer förmlich von der Aussonderung solcher Codes »lebt«.

4.5 Löschen ohne Verluste

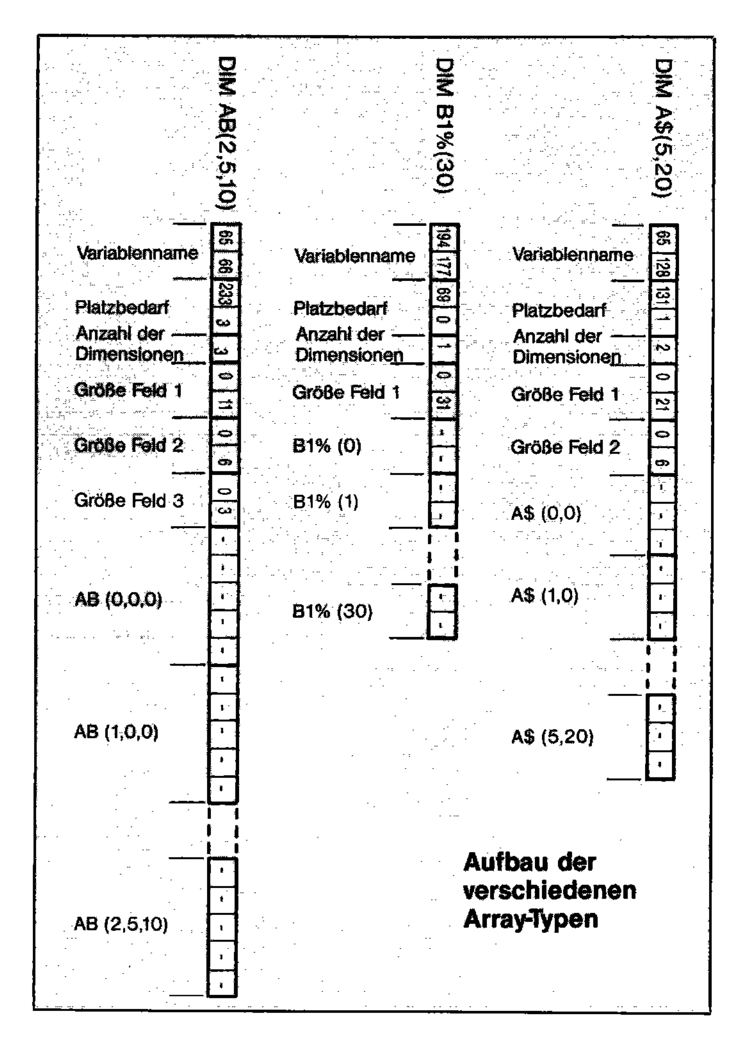


Bild 4.1: Aufbau der verschiedenen Array-Typen

Ein dimensioniertes Feld läßt sich normalerweise nicht ohne Verlust aller (!) Variablenwerte aus dem Speicher entfernen; das Programm »CLEAR-DIM« macht's möglich. Es wird über

```
LOAD"CLEAR-DIM",8,1
```

geladen, worauf man zunächst NEW eingeben sollte. Durch SYS49152 wird dann der CLR-Befehl erweitert. Folgt nun nach CLR ein Variablenname (Beispiel: CLR A$), so wird diese Variable, sofern sie existiert, aus dem Array-Bereich entfernt. »CLR« für sich genommen arbeitet wie gewohnt.
Einiges über den Aufbau der einzelnen Arrays und Variablentypen (siehe »C64 für Insider«) zeigt Bild 4.1.

4.6 Acht Mini-Tools

In diesem Unterkapitel erhalten Sie acht kleine, aber äußerst leistungsfähige Maschinenroutinen, die sich direkt für Basic-Zwecke anbieten. Sie belegen insgesamt den Bereich $C000-$C6AE, sind aber jedes auch für sich alleine lauffähig. Bei Bedarf laden Sie dann ein entsprechendes Programm über LOAD"NAME",8,1; beim Laden im Direktmodus ist anschließend NEW erforderlich.

Tool #1: PAUSE

Format: `SYS 49152,sekunden*60`
Das laufende Programm wird für eine definierte Zeitspanne (»seconds«) angehalten. Beispiel: SYS 49152,102 entspricht einer Pause von 1.75 Sekunden.

Tool #2: UNDERROMREAD

Format: `SYS 49200,adresse`
Dieses Programm dient zum Auslesen von Speicherstellen »unter« dem Basic-Interpreter oder dem Betriebssystem. In Basic können diese nämlich nicht ausgelesen werden, weil hierzu der Interpreter abgeschaltet werden muß. SYS 49200,60000 liest Speicherzelle 60000 unter dem Betriebssystem-ROM aus und übergibt den Wert an Adresse 2.

Tool #3: ARRAYSEEK

Format: `SYS 49240,arrayname,suchtext$`
Ein eindimensionales Stringarray wird nach einem bestimmten String durchsucht. Die Stringnummer steht nachher in den Adressen 176/177. Wurde der String nicht gefunden, steht hier 0.
`SYS 49240,MI,"64'ER"` durchsucht das Array MI$ nach dem String "64'ER".

Tool #4: SCREEN-TOOL

Screen-Tool umfaßt einige Routinen, die das Arbeiten mit dem Bildschirm etwas erleichtern.

1. Cursor setzen:
   `SYS 49500, zeile, spalte`
2. Zeilen löschen:
   `SYS 49503, erste_zeile, letzte_zeile`
   (schneller als Einzeiler #21)
3. Bildschirmteil invertieren:
   `SYS 49506,zeile,spalte,anzahl_der_zeichen`
4. Bildschirmteil re-invertieren:
   `SYS 49521,zeile,spalte,anzahl_der_zeichen`

Entgegen der normalen Betriebssystempraxis beginnt Screen-Tool bei der Numerierung der Zeilen bzw. Spalten nicht mit 0, sondern mit 1. Dies soll für den Basic-Anwender ein Entgegenkommen sein. Bei »anzahl_der_zeichen« ist jeweils 255 die Obergrenze.

Tool #5: SPRITESET

Format: `SYS 49700,sprite_nummer,block_nummer,x_pos,y_pos,farbe,expand`
Spriteset setzt die Parameter eines Sprites und ist ein weiterer Schritt weg von PEEK&POKE-Orgien. Beispiel: SYS 49700,0,11,100,50,6,1 setzt für Sprite Nr. 0 den Block 11 (ab Adresse 704) als Datenspeicher fest, positioniert ihn auf die Koordinaten 100/50, färbt ihn blau (Farbcode 6) und vergrößert ihn in X- und Y-Richtung (expand = 1).

Tool #6: GENERAL INPUT

In Kapitel 3 haben Sie bereits eine Basic-Eingaberoutine erhalten. Die Mini-Tools enthalten mit General Input ein weiteres Programm zu diesem Zweck. Daß es in Maschinensprache verfaßt ist, hat Vor- und Nachteile:

- Es ist in ein Basic-Programm schwieriger zu integrieren als ein Basic-Modul.
- Änderungen lassen sich nur für Maschinensprachler vornehmen.

+ Dafür ist GENERAL INPUT aber besonders schnell.
+ GENERAL INPUT nimmt keinen Basic-Speicher weg.

Nun zur Anwendung:

Format: `SYS 50500,maximallänge_1_bis_79`
Der Text steht nachher im als erstes definierten String. In der ersten Programmzeile sollte also immer ein Leerstring definiert werden, etwa

```
0 IN$=""
```

Welche Zeichen bei der Eingabe akzeptiert werden, hängt ausschließlich von der Bitstellung im Kontrollregister, der Adresse #189, ab:

Bit 0 (Wert 1)	:	Buchstaben A–Z sind erlaubt
Bit 1 (Wert 2)	:	Ziffern 0–9 werden akzeptiert
Bit 2 (Wert 4)	:	Grafikzeichen dürfen eingegeben werden
Bit 3 (Wert 8)	:	Interpunktionszeichen (. ; , : + – * / etc.)
Bit 4 (Wert 16)	:	kontrolliert, ob der Eingabe-Cursor blinkt

Dabei ist auch das Aufaddieren der Werte zulässig, etwa

```
POKE 189,1+2+8
```

Während der Eingabe löscht <DEL> das letzte Zeichen, <HOME> die ganze Eingabe. Dabei wird ein Maskenteil, der sich eventuell rechts vom Eingabefeld befindet, nicht – wie zu erwarten wäre – nach links verschoben; dies wurde durch Verwendung der Sequenz

```
CHR$(32)CHR$(157)CHR$(157)CHR$(32)CHR$(157)
```

anstelle von CHR$(20) erreicht.
Noch ein wichtiger Hinweis: Da der eingegebene String nicht im regulären Stringspeicher abgelegt wird, sondern direkt nach dem GENERAL-INPUT-Maschinenprogramm, sollte man ihn nur auslesen, aber hierbei nicht MID$, RIGHT$ oder LEFT$ benutzen. Sollten diese Funktionen dennoch nötig sein, empfiehlt es sich, eine neuen String anzulegen, welcher dem alten gleich ist, aber aus Teilen des alten zusammengesetzt ist:

`X$=LEFT$(IN$,1)+MID$(IN$,3)` anstelle von

`IN$=LEFT$(IN$,1)+MID(IN$,3)`

Tool #7: MENUE

Format: `SYS 50000,"PUNKT1,PUNKT2,PUNKT3"`
Der Bildschirm zeigt ein Menü mit den Menüpunkten, die im Parameterstring – durch Kommata getrennt – angegeben wurden. Rechts vom ersten Punkt steht ein Pfeil, der mit dem Joystick in Port #2 bewegt werden kann. Ein Druck auf den Feuerknopf übernimmt einen Menüpunkt. Dessen Nummer steht nachher in Adresse 702.
SYS 50000,"EINGABE,AUSGABE,DISKETTE,DIENST,ENDE" wäre zum Beispiel für eine Dateiverwaltung zu gebrauchen.

Tool #8: DISK-TOOL

Disk-Tool umfaßt einige Routinen, die den Umgang mit der Floppy erleichtern.

1. Programm absolut speichern:
   `SYS 50250,"name",startadresse,endadresse`

2. Programm absolut laden:
   `SYS 50253,"name",startadresse`

3. Disk-Kommando senden:
   `SYS 50256,"kommando"`
   Anschließend wird automatisch der Disk-Status ausgelesen.

4. Disk-Status auslesen:
   `SYS 50256,""`
   Dies entspricht Punkt 3, verzichtet aber auf ein Kommando.

4.7 Ein Dutzend Farbeffekte

Einzeiler #45 vermochte bereits ansatzweise zu demonstrieren, welche tollen Effekte durch blitzschnelles Umschalten der Bildschirmfarben zustandekommen. Hat Ihnen bereits dieser Basic-Ansatz gefallen, so bleibt Ihnen bei Betrachtung der Programme »FARBEFFEKT 1« bis »FARBEFFEKT 12« sicher die Spucke weg!
Laden Sie diese Programme wie gewöhnliche Basic-Programme und starten Sie sie mit RUN. Die ersten 11 Effekte werden durch <SHIFT>, <CBM>, <SPACE> oder eine Bewegung des Joysticks in Port 1 abgebrochen, um sogleich das nächste Programm zu laden. Bei »FARBEFFEKT 12« genügt das Drücken einer beliebigen Taste.
Alle Maschinenroutinen liegen im Bereich ab $C000 (#49152). Die Basic-Ladeprogramme sind ohne Mühe an andere Zeilennummern anzupassen. In manchen REM-Zeilen finden Sie noch Hinweise zum geringfügigen Modifizieren des jeweiligen Effektes, wodurch sich unzählige interessante Kombinationen ergeben.
Nun zur Funktionsweise: Ausgehend von einem einfachen Grundprogramm bringen bereits minimale Veränderungen einzelner Befehle und Verzögerungen völlig neue Bilder. Herausgegriffen seien die Farbeffekte 1, 2 und 7 für Maschinensprache-Interessierte.
Da im wesentlichen nur Zugriffe auf die VIC-Register sowie Verzögerungsschleifen vorliegen, sind die Kommentare kurz gefaßt. Wichtig: Mit »Bildschirm-Blanking« ist das Löschen von Bit 4 in VIC-Register #17 gemeint; dadurch wird der gesamte Bildschirm in der Farbe aus Adresse $d020 (Rahmenfarbe) dargestellt, eventueller Text ist nicht mehr sichtbar. Der Schnelligkeit unserer Effekte kommt dies sehr zugute, weil jeweils nur Adresse $d021 zu beeinflussen ist.

Listing 4.4: Disassembler-Listings zu den Farbeffekten 1, 2 und 7

Farbeffekt 1:

```
. C000  78         SEI          ; IRQ aus

. C001  AD 20 D0 LDA $D020    ; alte Rahmenfarbe
. C004  48         PHA          ; retten

. C005  AD 11 D0 LDA $D011    ; Bildschirm-
. C008  29 EF      AND #$EF     ; Blanking
. C00A  8D 11 D0 STA $D011    ; einschalten

. C00D  A2 00      LDX #$00     ; Zähler initialisieren

. C00F  8E 20 D0 STX $D020    ; Bildschirmfarbe setzen

. C012  AD 01 DC LDA $DC01    ; Taste gedrückt oder
. C015  C9 FF      CMP #$FF     ; Joystick bewegt?
. C017  D0 0D      BNE $C026    ; ja (Z=0): Abbruch
. C019  E8         INX          ; Zähler erhöhen
. C01A  EA         NOP          ; Verzögerungs-
. C01B  EA         NOP          ; schleife
. C01C  EA         NOP          ; für jeweils 10
. C01D  EA         NOP          ; Takt-
. C01E  EA         NOP          ; zyklen


. C01F  E0 02      CPX #$02     ; Zähler schon auf 2?
. C021  D0 EC      BNE $C00F    ; nein (Z=0): nicht initialisieren
. C023  4C 0D C0 JMP $C00D    ; Zähler initialisieren und weiter
-------------------------------

; Abbruch bei Tastendruck oder Joystickbewegung:

. C026  AD 11 D0 LDA $D011    ; Bildschirm-
. C029  09 10      ORA #$10     ; Blanking
. C02B  8D 11 D0 STA $D011    ; ausschalten

. C02E  68         PLA          ; alte Rahmenfarbe
. C02F  8D 20 D0 STA $D020    ; wiederherstellen
```

```
. C032  58          CLI          ; Interrupt zulassen
. C033  60          RTS          ; zurück ins Basic
------------------------------
```

Farbeffekt 2:

```
. C000  78          SEI          ; IRQ aus

. C001  AD 20 D0 LDA $D020    ; alte Rahmenfarbe
. C004  48          PHA          ; retten

. C005  AD 11 D0 LDA $D011    ; Bildschirm-
. C008  29 EF       AND #$EF     ; Blanking
. C00A  8D 11 D0 STA $D011    ; einschalten

. C00D  A2 00       LDX #$00     ; Zähler initialisieren

. C00F  8E 20 D0 STX $D020    ; Bildschirmfarbe setzen

. C012  AD 01 DC LDA $DC01    ; Taste gedrückt oder
. C015  C9 FF       CMP #$FF     ; Joystick bewegt?
. C017  D0 0C       BNE $C025    ; ja (Z=0): Abbruch
. C019  E8          INX          ; Zähler erhöhen

. C01A  EA          NOP          ; Verzögerungs-
. C01B  EA          NOP          ; schleife für
. C01C  EA          NOP          ; jeweils 8
. C01D  EA          NOP          ; Taktzyklen

. C01E  E0 03       CPX #$03     ; Zähler schon auf 3?
. C020  D0 ED       BNE $C00F    ; nein (Z=0): nicht initialisieren
. C022  4C 0D C0 JMP $C00D    ; weiter, vorher Zähler wieder auf 0
------------------------------

; Abbruch bei Tastendruck oder Joystickbewegung:

. C025  AD 11 D0 LDA $D011    ; Bildschirm-
. C028  09 10       ORA #$10     ; Blanking
. C02A  8D 11 D0 STA $D011    ; ausschalten
. C02D  68          PLA          ; alte Rahmenfarbe
. C02E  8D 20 D0 STA $D020    ; wiederherstellen
```

```
. C031  58          CLI         ; Interrupt zulassen
. C032  60          RTS         ; zurück ins Basic
--------------------------------
```

Farbeffekt 7:

```
. C000  78          SEI         ; IRQ aus

. C001  AD 20 D0 LDA $D020   ; alte Rahmenfarbe
. C004  48          PHA         ; retten

. C005  AD 11 D0 LDA $D011   ; Bildschirm-
. C008  29 EF       AND #$EF    ; Blanking
. C00A  8D 11 D0 STA $D011   ; einschalten

. C00D  AD 01 DC LDA $DC01   ; Taste gedrückt oder
. C010  C9 FF       CMP #$FF    ; Joystick bewegt?
. C012  D0 28       BNE $C03C   ; ja (Z=0): Abbruch

. C014  A9 00       LDA #$00    ; erste Farbe
. C016  8D 20 D0 STA $D020   ; setzen
. C019  A9 02       LDA #$02    ; zweite Farbe
. C01B  8D 20 D0 STA $D020   ; setzen
. C01E  A9 03       LDA #$03    ; dritte Farbe
. C020  8D 20 D0 STA $D020   ; setzen
. C023  A9 05       LDA #$05    ; vierte Farbe
. C025  8D 20 D0 STA $D020   ; setzen
. C028  A9 06       LDA #$06    ; fünfte Farbe
. C02A  8D 20 D0 STA $D020   ; setzen
. C02D  A9 07       LDA #$07    ; sechste Farbe
. C02F  8D 20 D0 STA $D020   ; setzen

. C032  EA          NOP         ; kurze Verzögerung

. C033  E6 02       INC $02     ; Hilfszähler
. C035  E6 02       INC $02     ; zweimal erhöhen
. C037  A5 02       LDA $02     ; und auslesen
. C039  4C 0D C0 JMP $C00D   ; weiter in Schleife
--------------------------------
```

```
; Abbruch bei Tastendruck oder Joystickbewegung:

. C03C  AD 11 D0 LDA $D011  ; Bildschirm-
. C03F  09 10    ORA #$10   ; Blanking
. C041  8D 11 D0 STA $D011  ; ausschalten

. C044  68       PLA        ; alte Rahmenfarbe
. C045  8D 20 D0 STA $D020  ; wiederherstellen

. C048  58       CLI        ; Interrupt zulassen
. C049  60       RTS        ; zurück ins Basic
-------------------------------
```

5 Professionelle Programmierumgebung oder: Computern mit Komfort

Die bisherigen – und noch folgenden – Abschnitte befassen sich mit Techniken, welche Ihnen bei der Realisierung eigener Programmkonzepte mit Rat und Tat zur Seite stehen. In diesem Kapitel hingegen erhalten Sie alles das, was man als »Programmierumgebung« bezeichnet: Tools für die Entwicklung effizienter Basic-Programme. Diese erleichtern die Programmentwicklung in allen Bereichen; indirekt erhöht sich damit auch die Qualität des Ergebnisses, welches Sie beim Programmieren erzielen.
Natürlich gibt es auch hierfür bereits Hunderte, wenn nicht gar Tausende von entsprechenden C64-Programmen, die schon veröffentlicht wurden; hier erhalten Sie jedoch meiner Meinung nach mit »Paradoxon Basic« das kompromißlos wertvollste »Development Toolkit«, das man sich nur vorstellen kann. Lassen Sie sich aber auch von den anderen Programmen überraschen, die es ebenfalls »in sich« haben.

5.1 Welt-Rek-OLD!

Jede Computerzeitschrift hat mindestens schon drei OLD-Routinen und Dutzende von Programmen, welche diese Funktion beinhalten, vorgestellt. Auch in diesem Buch tauchte diese Befehlserweiterung schon in Form von Einzeiler #2 auf. Dennoch wage ich zu behaupten, daß keine OLD-Routine für den C64 auch nur annähernd so genial gelöst ist wie diejenige, welche Sie von der Masterdiskette über LOAD "OLD",8,1 laden. Möchten Sie den Startbefehl wissen? Sorry, den gibt es nicht: Laden genügt bereits, und schon ist ein gelöschtes Programm wiederhergestellt!
Da sich die OLD-Routine jedoch nicht zu verstecken braucht, stellt sie ganz dezent die Bildschirmfarben um: blauer Rahmen, hellgrauer Hintergrund und schwarze Schriftfarbe. Nun ist natürlich ein Autostart nichts mehr, was C64-Besitzer in ekstatische Zustände versetzt. Deshalb der eigentliche Grund dafür, weshalb die Überschrift von einem »Welt-Rekord« spricht:
Vom gesamten Basic-Speicher wird kein einziges Byte benötigt! Selbst Kassettenpuffer, $C000-Bereich und das gesamte RAM bleiben unangetastet. Der Trick dabei ist, daß die eigentliche OLD-Routine in den Basic-Eingabepuffer eingelesen wird, welcher ohnehin bei Eingabe des nächstbesten Befehls überschrieben wird; vom bleibenden Speicher wird kein einziges Byte benötigt. Der Autostart wird über Stapelmanipulation erreicht, was nicht nur absolut sicher und absturzfrei funktioniert, sondern auch die Basic-Vektoren von Autostart-Aktionen »verschont«.
Die OLD-Routine begeht also eine gewisse Gratwanderung: In ihrem Speicherbereich kann sie

sich nur bis zum Ende ihrer Ausführung aufhalten; beim nächstem Gebrauch ist sie erneut einzulesen. Dennoch ist logisch sichergestellt, daß das Programm einwandfrei funktioniert. Sogar mit den meisten Basic-Erweiterungen oder sonstigen Utilities kooperiert dieses sensationelle OLD vorbildlich.
Der eigentliche OLD-Algorithmus ist exakt derselbe wie in Einzeiler #2, diesmal allerdings in Maschinensprache formuliert:

Listing 5.1: Quelltext OLD.SRC

```
100  -: ******************************
110  -: *                            *
120  -: * DAS RAFFINIERTESTE >>OLD<< *
130  -: *                            *
140  -: *   FUER DEN COMMODORE 64    *
150  -: *                            *
160  -: ******************************
170  -: *                            *
180  -: * SPEICHERPLATZBEDARF: 0.0    *
190  -: *                            *
200  -: *   REALISIERT DURCH STAPEL- *
210  -: *   MANIPULATION, AUTOSTART  *
220  -: *   UND SYSTEMEINGABEPUFFER  *
230  -: *                            *
240  -: ******************************
250  -: *                            *
260  -: * (C) 1985,1986,1987 BY      *
270  -: *                            *
280  -: *     FLORIAN MUELLER        *
290  -: *                            *
300  -: ******************************
310  -:
320  -:
330  -.BA $0200 ; NICHT ASSEMBLIEREN!
340  -;           NUR ZUR DOKUMENTATION
350  -;
360  -             LDY #1         ; OFFSET = 1
370  -             LDA 44         ; WERT HOLEN
380  -             STA (43),Y     ; AN BASICSTART
390  -;
400  -             JSR $A533      ; ZEILEN KOPPELN
410  -;
420  -             LDA 35         ; ZEIGER
430  -             STA 46         ; AUF
440  -             CPX #254       ; PROGRAMM-
450  -             BCC ERHOEHEN   ; ENDE
460  -             INC 46         ; BERECHNEN
470  -ERHOEHEN     INX            ; UND
480  -             INX            ; GLEICH
490  -             STX 45         ; SETZEN
500  -;
510  -             JSR $A663      ; CLR-BEFEHL
520  -             JMP $A474      ; "READY."
```

5.2 Pfund-ig!

Mit dem Symbol für das englische Pfund als Filenamen wird eine kleine Hilfsroutine zur Diskettenansteuerung bezeichnet. Sie ist gewissermaßen mit dem Utility »DOS 5.1« von der Test-/Demo-Diskette, die Commodore mit der Floppy 1541 ausliefert, zu vergleichen. Geladen wird »<£>« über LOAD"<£>",8,1, woraufhin die eigentliche Routine »DOS-HELP $C000« geladen und aktiviert wird.

Die Befehlserweiterung ist aus Anwendersicht relativ uninteressant; sie erweitert das Basic 2.0 um mehrere Befehle, welche mit dem Klammeraffensymbol (Taste rechts von <P>) eingeleitet werden:

<@>$	Directory-Anzeige
<@>C	Übergabe eines Befehlsstrings wie »I«
<@>E	Auslesen des Fehlerkanals
<@>L	Laden eines Programms
<@>S	Speichern eines Programms
<@>V	Verifizieren eines Programms
<@>M	Anhängen (Mergen) eines Programms
<@>A	Ändern der Geräteadresse (von 8 auf 9)
<@>Q	Verlassen der Erweiterung (Quit)
<@>R	Laden an eine relative Adresse → Syntax: <@>R"NAME",adresse

Das Prädikat »wertvoll« ist jedoch für den dokumentierten Quelltext angebracht:

Listing 5.2: Quelltext zu DOS-HELP $C000

```
formatiertes listing  teil 1

110:   c000                    *=   49152
115:   c000                    .opt p4,oo
120:                   ;
130:                   ; symbols
140:                   ;
145:   0073            chrget  =    $73
146:   0079            chrgot  =    $79
150:   a7e7            bef.alt =    $a7e7
160:   a7ae            inter   =    $a7ae
170:   af08            synerr  =    $af08
180:   0308            befehl  =    $308
190:   able            strout  =    $able
200:                   ;
205:   c000 d8                 cld
210:   c001 a9 18              lda  #<newvec
220:   c003 8d 08 03           sta  befehl
230:   c006 a9 c0              lda  #>newvec
240:   c008 8d 09 03           sta  befehl+1
250:   c00b a9 08              lda  #8
260:   c00d 8d 17 c0           sta  device
```

```
270:   c010 a9 cd              lda   #<text1
280:   c012 a0 c1              ldy   #>text1
290:   c014 4c 1e ab           jmp   strout
300:   c017          device    *=    *+1
310:   c018 20 73 00 newvec    jsr   chrget
320:   c01b c9 40              cmp   #"@"
330:   c01d f0 06              beq   newbef
340:   c01f 20 79 00           jsr   chrgot
350:   c022 4c e7 a7           jmp   bef.alt
360:   c025 20 73 00 newbef    jsr   chrget
370:   c028 a2 0b              ldx   #tbend-table
380:   c02a dd 41 c2 loop1     cmp   table-1,x
390:   c02d f0 06              beq   found
400:   c02f ca                 dex
410:   c030 d0 f8              bne   loop1
420:   c032 4c 08 af           jmp   synerr
430:   c035 ca       found     dex
440:   c036 bd 4d c2           lda   loadd,x
450:   c039 8d 43 c0           sta   sprung+1
460:   c03c bd 58 c2           lda   hiadd,x
470:   c03f 8d 44 c0           sta   sprung+2
480:   c042 4c 00 00 sprung    jmp   0          ; wird modifiziert
1000:                ;
1010:                ; ************ directory
1020:                ;
1030:  c045 a9 24    dir       lda   #"$"
1040:  c047 85 fb              sta   $fb
1050:  c049 a9 fb              lda   #$fb
1060:  c04b 85 bb              sta   $bb
1070:  c04d a2 00              ldx   #0
1080:  c04f 86 bc              stx   $bc
1090:  c051 e8                 inx
1100:  c052 86 b7              stx   $b7
1110:  c054 ad 17 c0           lda   device
1120:  c057 85 ba              sta   $ba
1130:  c059 a9 60              lda   #$60
1140:  c05b 85 b9              sta   $b9
1150:  c05d 20 d5 f3           jsr   $f3d5
```

formatiertes listing teil 2

```
1160:  c060 a5 ba              lda   $ba
1170:  c062 20 b4 ff           jsr   $ffb4
1180:  c065 a5 b9              lda   $b9
1190:  c067 20 96 ff           jsr   $ff96
1200:  c06a a9 00              lda   #0
1210:  c06c 85 90              sta   $90
1220:  c06e a0 03              ldy   #3
1230:  c070 84 fb    dir1      sty   $fb
1240:  c072 20 a5 ff           jsr   $ffa5
1250:  c075 85 fc              sta   $fc
1260:  c077 a4 90              ldy   $90
1270:  c079 d0 2f              bne   dir4
1280:  c07b 20 a5 ff           jsr   $ffa5
1290:  c07e a4 90              ldy   $90
1300:  c080 d0 28              bne   dir4
1310:  c082 a4 fb              ldy   $fb
1320:  c084 88                 dey
1330:  c085 d0 e9              bne   dir1
1340:  c087 a6 fc              ldx   $fc
1350:  c089 20 cd bd           jsr   $bdcd
1360:  c08c a9 20              lda   #" "
1370:  c08e 20 d2 ff           jsr   $ffd2
```

```
1380:  c091 20 a5 ff dir3     jsr  $ffa5
1390:  c094 a6 90             ldx  $90
1400:  c096 d0 12             bne  dir4
1410:  c098 aa                tax
1420:  c099 f0 06             beq  dir2
1430:  c09b 20 d2 ff          jsr  $ffd2
1440:  c09e 4c 91 c0          jmp  dir3
1450:  c0a1 a9 0d    dir2     lda  #13
1460:  c0a3 20 d2 ff          jsr  $ffd2
1470:  c0a6 a0 02             ldy  #2
1480:  c0a8 d0 c6             bne  dir1
1490:  c0aa 20 42 f6 dir4     jsr  $f642
1500:  c0ad 20 73 00          jsr  chrget
1500:  c0b0 4c ae a7          jmp  inter
1510:                ;
1520:                ;
2000:  c0b3 20 73 00 dco      jsr  chrget  ; ****** diskbefehl
2005:  c0b6 20 d4 e1          jsr  $e1d4
2010:  c0b9 a0 6f             ldy  #$6f
2020:  c0bb 20 b7 ab          jsr  $abb7
2030:  c0be ae 17 c0          ldx  device
2040:  c0c1 20 02 fe          jsr  $fe02
2050:  c0c4 20 d5 f3          jsr  $f3d5
2060:  c0c7 4c ae a7          jmp  inter
3000:  c0ca 20 73 00 dst      jsr  chrget  ; ****** diskstatus
3010:  c0cd ad 17 c0          lda  device
3020:  c0d0 85 ba             sta  $ba
3030:  c0d2 20 b4 ff          jsr  $ffb4
3040:  c0d5 a9 6f             lda  #$6f
3050:  c0d7 85 b9             sta  $b9
3060:  c0d9 20 96 ff          jsr  $ff96
3070:  c0dc 20 a5 ff dst1     jsr  $ffa5
3080:  c0df c9 0d             cmp  #13
3090:  c0e1 f0 06             beq  dst2
3100:  c0e3 20 16 e7          jsr  $e716
3110:  c0e6 4c dc c0          jmp  dst1
```

formatiertes listing teil 3

```
3120:  c0e9 20 16 e7 dst2     jsr  $e716
3130:  c0ec 20 ab ff          jsr  $ffab
3140:  c0ef 4c ae a7          jmp  inter
4000:  c0f2 20 73 00 dld      jsr  chrget  ; ******* diskload
4010:  c0f5 ae 17 c0          ldx  device
4020:  c0f8 a0 00             ldy  #0
4030:  c0fa 98                tya
4040:  c0fb 20 ba ff          jsr  $ffba
4050:  c0fe 20 57 e2          jsr  $e257   ; holt filenamen
4060:  c101 a9 00             lda  #0
4070:  c103 85 0a             sta  10
4080:  c105 20 6f e1          jsr  $e16f
4090:  c108 4c ae a7          jmp  inter
5000:  c10b 20 73 00 dsv      jsr  chrget  ; ******* disksave
5010:  c10e ae 17 c0          ldx  device
5020:  c111 a0 00             ldy  #0
5030:  c113 98                tya
5040:  c114 20 ba ff          jsr  $ffba
5050:  c117 20 57 e2          jsr  $e257
5060:  c11a a6 2d             ldx  $2d
5070:  c11c a4 2e             ldy  $2e
5080:  c11e a9 2b             lda  #$2b
5090:  c120 20 d8 ff          jsr  $ffd8
5100:  c123 90 03             bcc  dsv1
```

```
5110:   c125 4c f9 e0           jmp  $e0f9    ; fehlerbehandlung
5120:   c128 4c ae a7 dsv1      jmp  inter
6000:   c12b 20 73 00 dve       jsr  chrget   ; ************* diskverify
6010:   c12e ae 17 c0           ldx  device
6020:   c131 a0 00              ldy  #0
6030:   c133 98                 tya
6040:   c134 20 ba ff           jsr  $ffba
6050:   c137 20 57 e2           jsr  $e257
6060:   c13a a9 01              lda  #1
6070:   c13c 85 0a              sta  10
6080:   c13e 20 6f e1           jsr  $e16f
6090:   c141 4c ae a7           jmp  inter
7000:   c144 20 73 00 dme       jsr  chrget   ; ******* diskmerge
7010:   c147 a9 00              lda  #0
7020:   c149 85 0a              sta  10
7030:   c14b ae 17 c0           ldx  device
7040:   c14e a8                 tay
7050:   c14f 20 ba ff           jsr  $ffba
7060:   c152 20 57 e2           jsr  $e257
7070:   c155 a5 2d              lda  $2d
7080:   c157 38                 sec
7090:   c158 e9 02              sbc  #2
7100:   c15a aa                 tax
7110:   c15b a5 2e              lda  $2e
7120:   c15d e9 00              sbc  #0
7130:   c15f a8                 tay
7140:   c160 ad 17 c0           lda  device
7150:   c163 85 ba              sta  $ba
7160:   c165 a5 0a              lda  10
7170:   c167 4c 75 e1           jmp  $e175
8000:   c16a 20 9b b7 dad       jsr  $b79b
8010:   c16d e0 08              cpx  #8
8020:   c16f b0 03              bcs  dad1
8030:   c171 4c 48 b2 illquan   jmp  $b248
```

formatiertes listing teil 4

```
8040:   c174 e0 0d    dad1      cpx  #13
8050:   c176 b0 f9              bcs  illquan
8060:   c178 8e 17 c0           stx  device
8070:   c17b 4c ae a7           jmp  inter
9000:   c17e 20 73 00 qui       jsr  chrget
9010:   c181 20 53 e4           jsr  $e453
9020:   c184 a9 22              lda  #<text2
9030:   c186 a0 c2              ldy  #>text2
9040:   c188 20 1e ab           jsr  strout
9050:   c18b 4c ae a7           jmp  inter
9100:   c18e 20 73 00 rel       jsr  chrget
9110:   c191 ae 17 c0           ldx  device
9120:   c194 a0 00              ldy  #0
9130:   c196 98                 tya
9140:   c197 20 ba ff           jsr  $ffba
9150:   c19a 20 57 e2           jsr  $e257
9160:   c19d 20 fd ae           jsr  $aefd
9170:   c1a0 20 8a ad           jsr  $ad8a
9180:   c1a3 20 f7 b7           jsr  $b7f7
9190:   c1a6 a9 00              lda  #0
9200:   c1a8 a6 14              ldx  $14
9210:   c1aa a4 15              ldy  $15
9220:   c1ac 20 d5 ff           jsr  $ffd5
9230:   c1af 4c ae a7           jmp  inter
9500:   c1b2 20 73 00 ori       jsr  chrget
9510:   c1b5 a9 01              lda  #1
```

```
9520:  c1b7 ae 17 c0          ldx  device
9530:  c1ba a8                tay
9540:  c1bb 20 ba ff          jsr  $ffba
9550:  c1be 20 57 e2          jsr  $e257
9560:  c1c1 a9 00             lda  #0
9570:  c1c3 20 d5 ff          jsr  $ffd5
9580:  c1c6 4c ae a7          jmp  inter
9610:  c1c9 a6 14             ldx  $14
9620:  c1cb a4 15             ldy  $15
20000:                 ;
20010:                 ; texte
20020:                 ;
20030: c1cd 0d 93 0d text1    .byte13,$93,13
20040: c1d0 43 4f 4d          .asc "commodore 64 - dos help 1541 version 1.0"
20050: c1f8 0d                .byte13
20060: c1f9 28 43 29          .asc "(c) 1985 by florian mueller  $c000-$c263"
20070: c221 00                .byte0
20100: c222 0d       text2    .byte13
20110: c223 4e 45 55          .asc "neuaktivieren mit : sys 49152"
20120: c240 0d 00             .byte13,0
30000:                 ;
30010:                 ; befehlstabelle
30020:                 ;
30030:                 ;
30040: c242 24 43 45 table    .asc "$celsvmaqro"
30050: c24d          tbend    =    *
30060:                 ;
30070: c24d 45 b3 ca loadd    .byte<dir,<dco,<dst,<dld,<dsv,<dve,<dme
30080: c254 6a 7e 8e          .byte<dad,<qui,<rel,<ori
30090:                 ;
30100: c258 c0 c0 c0 hiadd    .byte>dir,>dco,>dst,>dld,>dsv,>dve,>dme,>ori
30110: c260 c1 c1 c1          .byte>dad,>qui,>rel


formatiertes listing  teil 5

30120:                 ;
]c000-c263



no errors
```

Da ich dieses Programm vor längerer Zeit verfaßt habe, wobei ich doch nicht mit Hypra-Ass arbeiten konnte, habe ich den Quelltext nicht auf der Masterdiskette abgelegt. Eine direkte Modifikation dieses Programms ist auch nicht das Entscheidende, sondern Sie sollen daraus lernen, wie man auf schnörkellose Weise eigene Befehle integriert.

5.3 Tool-Creat-iv!

Manche Utilities braucht man einfach auf jeder Arbeitsdiskette. Deshalb habe ich vor längerer Zeit ein Programm namens »Tool-Creator« verfaßt, welches die wichtigsten Utilities auf einer Diskette generiert. Es wird mit LOAD "TC",8,1 geladen; dieser Autostart-Vorspann lädt sofort das Hauptprogramm TOOL-CREATOR nach.
Dieses schließlich meldet sich zunächst mit einem monströsen Einschaltbild, welches nach einem gemessenen Zeitabschnitt verschwindet; während der »Sendepause« werden alle Daten eingelesen sowie Prüfungen vorgenommen, ob das Programm unbefugterweise modifiziert wurde. Anschließend erscheint das komfortable Hauptmenü, welches über Cursortasten, <F1> und <RETURN> (gleiche Wirkung wie <F1>) gesteuert wird:

Directory anzeigen
Disk-Befehl ausführen
Hilfsprogramm generieren
Autostart-Lader generieren

Die ersten beiden Menüpunkte bedürfen keiner weiteren Erläuterung; sie sind übliche Dienstfunktionen, welche Auswahl und Vorbereitung einer Zieldiskette für die generierten Tools so komfortabel wie möglich gestalten wollen. Die beiden anderen Optionen sind jedoch etwas komplexer.

Hilfsprogramm generieren

Dazu wählen Sie zunächst eine Diskette aus, auf welche die Tools geschrieben werden sollen. Drücken Sie dann, gemäß der Anweisung des Programms, die Taste <SHIFT>. Zunächst wird geprüft, ob die eingelegte Diskette formatiert ist. Bei positiver Rückmeldung der Floppy erscheint ein Selektionsmenü für die Tools <←>, OLD, <£>, DOS HELP $C000. Mit den Cursortasten bewegen Sie den Auswahlpfeil; durch Drücken von »n« für »Nein« oder »j« für »Ja« wählen Sie dann aus, welche Dateien auf die Zieldiskette gehören (»ja«) und welche nicht (»nein«). Ist Ihre Auswahl korrekt, löst <F1> den Schreibvorgang aus. Drei dieser Dateien befinden sich bereits auf der Programmdiskette:
OLD holt ein gelöschtes Programm zurück, <£> ist ein Autostart-Lader für DOS HELP $C000, welches wiederum Diskettenbefehle zur Verfügung stellt. Der <←> (nicht direkt auf der Programmdiskette enthalten, aber im Tool-Creator-Menü angeboten) bewirkt, daß das Directory der eingelegten Diskette angezeigt wird; ein Basic-Programm im Speicher wird zerstört, kann aber mit OLD zurückgeholt werden. Nach einmaligem Laden der Autostart-Datei <←> genügt SYS 828 zur Anzeige eines Disketten-Inhaltsverzeichnisses.
Ich weise noch einmal darauf hin, daß anstelle dieser Tools – sieht man vom Welt-Rek-OLD ab – neuere und bessere Problemlösungen an anderer Stelle stehen.

Autostart-Lader generieren

Damit können Sie zu einem beliebigen Basic-Programm, welches über LOAD" <name> ",8 geladen und mit RUN gestartet wird, eine 2 Blöcke lange Datei schaffen, die das Basic-Programm nachlädt und automatisch startet. Dies erspart nicht nur Tipparbeit (LOAD" <name> ",8,1 ist schneller eingegeben als LOAD" <name> ",8 und RUN), sondern wirkt vor allem sehr professionell. Sie geben Tool-Creator den Namen der nachzuladenden Basic-Datei an sowie den Filenamen, unter welchem Sie den Autostart-Lader vorfinden möchten. Den Autostart-Lader dürfen Sie im nachhinein noch umbenennen, die eigentliche Basic-Datei hingegen nicht (sonst wirkt der Autostart nicht).
In Kapitel 11 erhalten Sie ein Generatorprogramm für einen etwas anders angelegten Autostart.

Das Programm »Tool-Creator«

Die Datei »TOOL-CREATOR« ist in der kompilierten Fassung auf Diskette abgelegt. Hier jedoch sei der Quelltext veröffentlicht (wenn auch nicht auf Diskette beigelegt), aus dem Sie sicher vieles über die Programmierung von Menüs, simplen Programmschutz, Floppy-Behandlung und Bildschirmmasken – kurz: professionelle Aufmachung – lernen können:

Listing 5.3: Quelltext zu TOOL-CREATOR

```
100 :
110 sys65409:poke788,49:poke792,193:poke793,254:poke789,234
120 poke53280,6:poke53281,15:clr
130 printchr$(13)"(clr)(swuc)(blk)(dish)(rvs) (off)      (rvs) (off)▨  ▧(rvs) (o
ff)▨  ▧(rvs) (off) (rvs)                         ";
140 print"   (off) (rvs)    (off) (rvs)    (off) (rvs) (off) (rvs)    (off) (rvs) (
off) (rvs)                      ";
150 print"(rvs)   (off) (rvs)    (off) (rvs)    (off) (rvs) (off) (rvs)    (off) (r
vs) (off) (rvs)                         ";
160 print"(rvs)   (off) (rvs)    (off) (rvs)    (off) (rvs) (off) (rvs)    (off) (r
vs) (off) (rvs)    (off)     (rvs)        ";
170 print"(rvs)   (off) (rvs)    (off) (rvs)    (off) (rvs) (off) (rvs)    (off) (r
vs) (off) (rvs)                         ";
180 print"   (off) (rvs)    (off) (rvs)    (off) (rvs) (off) (rvs)    (off) (rvs) (
off) (rvs)                      ";
190 print"   (off) (rvs)   ▧(off)    (rvs)▨ ▧(off)    (rvs)▨ (off)    ▧(rvs)
        ";
200 print"                                                  ";
210 print"                                                  ";
220 print" (off)▨  ▧(rvs) (off)  ▧(rvs) (off)    (rvs) (off)▨  ▧(rvs) (off)
   (rvs) (off)▨  ▧(rvs) (off)  ▧(rvs) ";
230 print"(rvs) (off) (rvs)   (off) (rvs) (off) (rvs)  (off) (rvs) (off) (rvs)
  (off) (rvs)   (off) (rvs)   (off) (rvs)   (off) (rvs) (off) (rvs
)  (off) (rvs) ";
240 print"(rvs) (off) (rvs)     (off) (rvs)  (off) (rvs) (off) (rvs)    (off) (r
vs)   (off) (rvs)   (off) (rvs)   (off) (rvs)   (off) (rvs) (off) (rvs)  (off) (
rvs) ";
250 print"(rvs) (off) (rvs)     (off)   (rvs)▨ (off)    (rvs) (off)     (rvs)
(off) (rvs)   (off) (rvs)   (off) (rvs) (off)   (rvs)▨ ";
260 print" (off) (rvs)     (off) (rvs) ▧(off)▧(rvs) (off) (rvs)    (off) (rvs)
 (off) (rvs)   (off) (rvs)   (off) (rvs)   (off) (rvs) (off) (rvs) ▧(off)▧(rvs)
";
270 print"(rvs) (off) (rvs)   (off) (rvs) (off) (rvs)  (off) (rvs) (off) (rvs)
  (off) (rvs)   (off) (rvs)   (off) (rvs)   (off) (rvs) (off) (rvs
)  (off) (rvs) ";
```

```
280 print"(rvs) ▒(off)   (rvs)▒ (off) (rvs)  (off) (rvs) (off)    (rvs) (off) (r
vs)   (off) (rvs)   (off) (rvs)  ▒(off)   (rvs)▒ (off) (rvs)  (off) (rvs) ";
290 print"(blu)                                              (blk)(off)";
300 print"U─────────────────────────────────I";
310 print"|copyright (c) 1985  by florian mueller|";
320 print"J─────────────────────────────────K";
330 print"(rvs)(blu)                                            (blk)(off)";
340 print"(down)     checking program - please wait";
350 rem datas fuer old einlesen
360 :
370 dim ol%(307)
380 ol%(0)=0:ol%(1)=1
390 for i = 2 to 260:ol%(i)=2:next i
400 for i = 1 to 47:read ol%(260+i):next i
410 data162,255,154,169,6,141,32,208,169,15,141,33,208,169,0,141,134,2,160
420 data1,165,44,145,43,32,51,165,165,35,133,46,224,254,144,2,230,46,232,232
430 data134,45,32,99,166,76,116,164
440 :
450 rem datas fuer ← einlesen
460 :
470 dim dr%(237)
480 dr%(0)=2:dr%(1)=3
490 for i=2to237:read dr%(i):next i
500 data60,3,124,165,26,167,228,167,134,174,199,0,64,240,76,72,178,0,49,234
510 data20,194,71,254,74,243,145,242,14,242,80,242,51,243,87,241,202,241,237
520 data246,62,241,47,243,102,254,165,244,237,245,0,0,0,0,0,0,0,169,131,162
530 data164,141,2,3,142,3,3,169,36,133,251,169,251,133,187,169,0,133,188,169
540 data1,133,183,169,8,133,186,169,96,133,185,32,213,243,165,186,32,180,255
550 data165,185,32,150,255,169,0,133,144,160,3,132,251,32,165,255,133,252,164
560 data144,208,58,32,165,255,164,144,208,51,164,251,136,208,233,166,252,32
570 data205,189,169,32,32,210,255,32,165,255,166,144,208,29,170,240,17,32,210
580 data255,173,197,0,201,64,240,4,201,60,240,245,76,146,3,169,13,32,210,255
590 data160,2,208,187,32,66,246,162,0,189,208,3,105,2,32,210,255,232,138,201
600 data30,208,242,162,128,76,55,164,0,0,11,16,38,65,39,30,47,55,54,51,30,64
610 data87,30,68,74,77,80,71,63,76,30,75,83,67,74,74,67,80,11
620 :
630 rem datas fuer £ lesen
640 :
650 dim bo%(320)
660 bo%(0)=0:bo%(1)=1
670 for i = 2 to 260:bo%(i)=2:next i
680 for i = 261 to 319:read bo%(i):next i
690 data162,255,154,169,1,168,162,8,32,186,255,169,14,162,48,160,2,32,189,255
700 data169,0,32,213,255,144,3,76,226,252,32,83,228,32,191,227,32,34,228,32
710 data0,192,76,134,227,68,79,83,32,72,69,76,80,32,36,67,48,48,48
720 :
730 rem datas fuer doshelp lesen
740 :
750 dim dh%(613)
760 dh%(0)=0:dh%(1)=192
770 for i = 2 to 613:read dh%(i):next
780 data216,169,24,141,8,3,169,192,141,9,3,169,8,141,23,192,169,205,160,193
790 data76,30,171,0,32,115,0,201,64,240,6,32,121,0,76,231,167,32,115,0,162
800 data11,221,65,194,240,6,202,208,248,76,8,175,202,189,77,194,141,67,192
810 data189,88,194,141,68,192,76,0,0,169,36,133,251,169,251,133,187,162,0,134
820 data188,232,134,183,173,23,192,133,186,169,96,133,185,32,213,243,165,186
830 data32,180,255,165,185,32,150,255,169,0,133,144,160,3,132,251,32,165,255
840 data133,252,164,144,208,47,32,165,255,164,144,208,40,164,251,136,208,233
850 data166,252,32,205,189,169,32,32,210,255,32,165,255,166,144,208,18,170
860 data240,6,32,210,255,76,145,192,169,13,32,210,255,160,2,208,198,32,66,246
870 data32,115,0,76,174,167,32,115,0,32,212,225,160,111,32,183,171,174,23,192
880 data32,2,254,32,213,243,76,174,167,32,115,0,173,23,192,133,186,32,180,255
890 data169,111,133,185,32,150,255,32,165,255,201,13,240,6,32,22,231,76,220
900 data192,32,22,231,32,171,255,76,174,167,32,115,0,174,23,192,160,0,152,32
910 data186,255,32,87,226,169,0,133,10,32,111,225,76,174,167,32,115,0,174,23
920 data192,160,0,152,32,186,255,32,87,226,166,45,164,46,169,43,32,216,255
930 data144,3,76,249,224,76,174,167,32,115,0,174,23,192,160,0,152,32,186,255
940 data32,87,226,169,1,133,10,32,111,225,76,174,167,32,115,0,169,0,133,10
```

```
2270 print"(down)Druecken Sie dann auf SHIFT ";:poke204,0:poke653,0:wait653,1
2280 poke207,0:poke204,1:print" "
2290 :
2300 printt$"(down)(down)     Autostartfile wird generiert"
2310 :
2320 :
2330 n$=l$+",p,w"
2340 open1,8,15,"i":close1
2350 open2,8,2,n$
2360 for i=1tolf
2370 al%(303+i)=asc(mid$(f$,i,1))
2380 next i
2390 lt% = 302+i
2400 for i=0tolt%:print#2,chr$(al%(i));:next i
2410 close 2
2420 printt$"(down)(down)     Autostartfile ist generiert."
2430 goto1660
2440 :
2450 :
2460 rem ***********************
2470 rem ***********************
2480 rem *** tool generieren ***
2490 rem ***********************
2500 rem ***********************
2510 :
2520 :
2530 printt$"(down)(down) Hilfsprogramme auf Diskette generieren"
2540 print" --------------------------------------(down)(down)(down)"
2550 fori=0to3:fg%(i)=-1:rv$(i)="":next:rv$(0)="(rvs)":s=0
2560 print"Bitte legen Sie die Diskette ein,auf der"
2570 print"sie die Hilfsprogramme wuenschen."
2580 print"(down)Druecken Sie dann SHIFT ";:poke204,0:poke653,0:wait653,1:poke20
7,0
2590 poke204,1:poke198,0
2600 gosub 2890:if ef then 1690
2610 printt$"(grn)(down)(down)(down) Auswahl ueber : Cursortasten, F1, j, n"
2620 print"---------------------------------------(blu)"
2630 sp$="  ":pf$="->"
2640 fori=0to3:r$(i)=sp$:sv(i)=-1:nexti
2650 s=0:r$(0)=pf$:goto2690
2660 ifsv(di)thenprint"ja  (down)"
2670 ifnot(sv(di))thenprint"nein(down)"
2680 return
2690 printchr$(13)"(home)(down)(down)(down)(down)(down)(down)"
2700 printr$(0)tab(10)chr$(34)"←"chr$(34)tab(30);:di=0:gosub2660
2710 printr$(1)tab(10)chr$(34)"old"chr$(34)tab(30);:di=1:gosub2660
2720 printr$(2)tab(10)chr$(34)"£"chr$(34)tab(30);:di=2:gosub2660
2730 printr$(3)tab(10)chr$(34)"DOS HELP $C000"chr$(34)tab(30);:di=3:gosub2660
2740 print"(grn)---------------------------------------(blu)"
2750 :
2760 wait198,1:geta$
2770 ifa$<>"(f1)"anda$<>"j"anda$<>"n"anda$<>"(down)"anda$<>"(up)"then2760
2780 ifa$="j"thensv(s)=-1:goto2690
2790 ifa$="n"thensv(s)=0:goto2690
2800 ifa$="(down)"thenr$(s)=sp$:s=3and(s+1):r$(s)=pf$:goto2690
2810 ifa$="(up)"thenr$(s)=sp$:s=3and(s-1):r$(s)=pf$:goto2690
2820 :
2830 rem **************auswahl getroffen
2840 rem **************---------------
2850 :
2860 goto 2960
2870 rem ***************fehlerbehandlung
2880 rem ***************---------------
2890 open1,8,15,"i0":close1:rem ********************init-test
2900 rem ***********einsprung fehlertest
2910 rem ***********------------------
2920 :
```

```
2930 open 1,8,15:input#1,a,a$,b,c:close1:ef=(a<>0)
2940 if not ef then return
2950 print"(down)(down)(red)Disk > "a" "a$" "b;c"(blu)":return
2960 :
2970 rem ********fortsetzung "tool gen."
2980 rem ********-----------------------
2990 printt$"(down)(down)(blu)"tab(15)"Bitte warten."
3000 if(notsv(0)andnotsv(1)andnotsv(2)andnotsv(3))then 1350
3010 gosub 2890
3020 if ef then 1690
3030 if not(sv(0)) then 3100
3040 print"(down)(down)(down)"chr$(34)"←"chr$(34)" wird generiert"
3050 gosub 2890
3060 if ef then 1690
3070 open 2,8,2,"←,p,w"
3080 for i=0to237:print#2,chr$(dr%(i));:next i
3090 close2:gosub2930
3100 if not(sv(1)) then 3170
3110 print"(down)(down)"chr$(34)"old"chr$(34)" wird generiert"
3120 gosub 2890
3130 if ef then 1690
3140 open2,8,2,"old,p,w"
3150 fori=0to307:print#2,chr$(ol%(i));:next
3160 close2:gosub2930
3170 if not(sv(2)) then 3230
3180 print"(down)(down)"chr$(34)"£"chr$(34)" wird generiert"
3190 gosub 2890
3200 open2,8,2,"£,p,w"
3210 fori=0to320:print#2,chr$(bo%(i));:next
3220 close2:gosub2930
3230 if not(sv(3)) then 3290
3240 print"(down)(down)"chr$(34)"dos help $c000"chr$(34)" wird generiert"
3250 gosub 2890
3260 open2,8,2,"dos help $c000,p,w"
3270 fori=0to613:print#2,chr$(dh%(i));:next
3280 close2:gosub2930
3290 print"(down)(down)(down)(rvs)********** Ende der Generation *********"
3300 goto1690
```

Der praktische Nutzen des Tool-Creator ist, nachdem seit seiner Entstehung viel Zeit verging, in der deutliche Fortschritte in der Programmierung des C 64 gemacht wurde, nicht mehr so groß wie 1985.

5.4 Paradoxon Basic oder: Das darf doch nicht wahr sein!

Ohne Übertreibung gibt es unter Garantie weltweit mehrere hundert Basic-Erweiterungen für den C64. Wieviel tausend Befehle diese insgesamt zur Verfügung stellen, weiß niemand mehr. Doch eines schaffte noch keine: 12 Kbyte mehr Speicherplatz für Basic zur Verfügung zu stellen, ohne sich einer Hardware-Erweiterung zu bedienen!
Wenn Sie jedoch

```
LOAD "PARADOXON-BASIC",8
```

und anschließend RUN eingeben, so erreichen Sie diesen Zustand!
Darauf haben schon viele gewartet: Eine Programmierhilfe, die vor allem das Schreiben umfangreicher Programme erleichtert und bereits existierende Befehle verbessert, um Ihnen umständliche Programmiertricks zu ersparen. Im Gegensatz zu Erweiterungen, die lediglich ein Plus an Befehlsreichtum bieten, liegt hierbei der unschätzbare Vorteil darin, daß höchstens kleinere Anpassungen erforderlich sind, um die mit Paradoxon Basic erstellten Programme an alle anderen C64-Anwender weiterzugeben. So gesehen, ist Paradoxon Basic weniger eine Erweiterung des Basic-Befehlsschatzes, als vielmehr ein Entwicklungssystem der uneingeschränkten Extra-Klasse.

5.4.1 Die Funktionsweise, erklärt für Speicher-Spezialisten

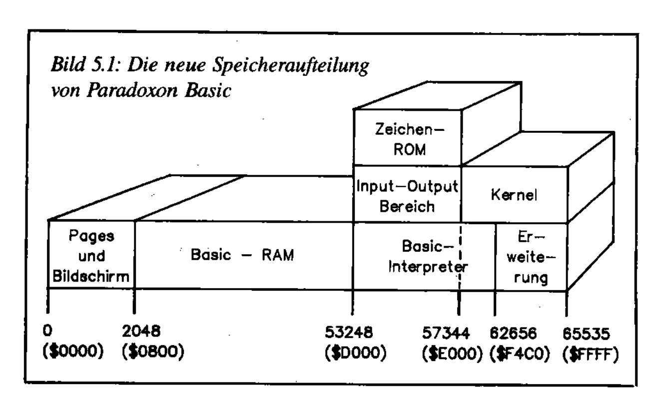


Bild 5.1: Die neue Speicheraufteilung von Paradoxon Basic

Der Programmname »Paradoxon Basic« (Filename: »PARADOXON-BASIC«) rührt daher, daß es paradox (griechisch: »gegen den Glauben, widersinnig«, Paradoxon = Widerspruch) vorkommt, wenn sich Basic und Erweiterung im RAM unter dem Kernel befinden. Normalerweise ist im C64 der Bereich von \$0800 (#2048) bis \$9FFF (#40959) als Basic-Speicher vorgesehen. Im Bereich von \$A000 (#40960) bis \$FFFF (#65535) bleiben so jedoch 24 Kbyte für Basic ungenutzt, da an diesen Stellen der Basic-Interpreter, der I/O-Bereich und das Kernel eingeblendet werden.
Der erweiterte Basic-Speicherplatz wird nun dadurch gewonnen, daß der Basic-Interpreter unter den I/O-Bereich und das Kernel, also ins RAM ab \$D000 (#53248), verschoben wird. Dabei bleiben noch rund 3 Kbyte frei, die Paradoxon Basic für die Befehlserweiterung und die neuen Umschaltroutinen nutzt.

Folgende Speicherbereiche müssen immer gleichzeitig zur Verfügung stehen:

1. Basic-RAM + Basic-Interpreter
   oder
2. Basic-RAM + I/O-Bereich + Kernel

Da der I/O-Bereich beim besten Willen nicht verschiebbar ist, diente er als Ansatzpunkt für die Speicheraufteilung. Der Basic-Interpreter mußte daher unter den I/O-Bereich und das Kernel:

1. Basic-RAM plus Basic-Interpreter: $0800–$CFFF, $D000–$F4C0
2. Basic-RAM plus I/O-Bereich plus Kernel: $0800–CFFF, $D000–$DFFF und $E000–$FFFF.

Da die Kernel-Routinen auch für die Arbeit des Basic-Interpreters essentiell sind, waren Umschaltroutinen notwendig, um zwischen den beiden Speicherkonfigurationen hin- und herzuschalten. Insgesamt werden vier Umschaltroutinen benötigt: Umschaltung zur Nutzung des Kernel sowie Interrupt-, NMI- und Reset-Behandlung.
Generell gilt: Die Umschaltung zwischen den beiden Speicheraufteilungen darf nur von einem Speicherbereich aus erfolgen, der selbst außerhalb des betroffenen Bereiches liegt, also beispielsweise aus ungenutzten Speicherstellen der ersten Pages. Die Speicherplätze $02A7–$02FF sind unbelegt und können dafür verwendet werden.
Alle vier Umschaltroutinen arbeiten nach gleichem Prinzip: Sie legen eine Rücksprungadresse auf den Stapel, schalten auf das Kernel um und starten die benötigte Routine. Der Rücksprung erfolgt nun auf die Umschaltroutine, welche wiederum ins RAM verzweigt. Der NMI wird ebenfalls ins RAM umgeleitet, wo auf <RUN/STOP> bzw. <CTRL C= RESTORE> getestet wird, um gegebenenfalls einen Warmstart oder einen Reset auszuführen. Die NMI- oder Reset-Routinen im Kernel können wegen der tiefgreifenden Umgestaltung des Basic-Speichers nicht mehr verwendet werden, sie würden abstürzen.
Da hinter den Umschaltroutinen und dem Basic-Interpreter noch etwa 2 Kbyte Speicherplatz übrigbleiben, wurde hier die neue Befehlserweiterung eingebaut. Sie verbessert auch vorhandene Befehle, enthält Diskettenkommandos, Programmierhilfen und eine Funktionstastenbelegung. Zusammen füllen diese Befehle den gesamten verbliebenen Speicher bis auf ein paar Byte aus.
So wird der gesamte C64-Speicher optimal genutzt und stolz die Einschaltmeldung verkündet:

```
51199 BASIC BYTES FREE
```

Die neue Speicheraufteilung finden Sie in Tabelle 5.1 noch einmal sehr detailliert.

$0000 - $07FF	(RAM)	erste Pages und Bildschirm
$02A7 - $02FF	(RAM)	Umschaltroutinen
$0800 - $CFFF	(RAM)	Basic-Speicherbereich (50 Kbyte)
$D000 - $FFFF	(ROM!)	I/O-Bereich und Kernel
$D000 - $F4BF	(RAM)	Basic-Interpreter (verschoben)
$F4C0 - $F5BA	(RAM)	eingebundene Befehle
$F5BB - $F6D0	(RAM)	Diskettenbefehle
$F6D2 - $FBFF	(RAM)	zusätzliche Befehle
$FC00 - $FCFF	(RAM)	Funktionstasten-Belegung
$FD00 - $FDD6	(RAM)	Funktionstasten-Routinen
$FDD7 - $FFFF	(RAM)	Umschaltroutinen

Tabelle 5.1: Die Speicheraufteilung von Paradoxon-Basic unterscheidet sich erheblich von der Normalkonfiguration

5.4.2 Umgang mit Paradoxon Basic

Die Datei »PARADOXON-BASIC« ist nur 14 Blöcke lang und somit schnell geladen. Nach dem Start über RUN dauert es knappe sieben Sekunden, bis das Programm generiert ist und seine Einschaltmeldung ausgibt. Neben dem erweiterten Basic-Speicherplatz von 50 Kbyte hat man eine Befehlserweiterung und Tool-Bibliothek zur Verfügung, die sich in mehrere Teile gliedert:
Die LOAD- und SAVE-Befehle verwenden automatisch die Geräteadresse 8 sowie die Sekundäradresse 1. Es wird also, sofern nicht ausdrücklich anders angegeben, immer absolut geladen (LOAD "NAME",8,1). Durch Drücken von <CTRL C= RESTORE> wird ein Reset ausgeführt, welcher Paradoxon Basic neu startet; der Befehl SYS 64738 ist nicht mehr verwendbar!
Die Funktionstasten <F1> bis <F8> können jeweils mit bis zu 31 Zeichen Text und Steuerzeichen belegt werden. Um sie zu programmieren, drückt man einfach <CTRL> zusammen mit der entsprechenden Funktionstaste. Daraufhin verdoppelt sich blitzartig die Blinkgeschwindigkeit des Cursors. Alle Tastatureingaben werden jetzt in den Speicher übernommen – auch Steuertasten! Die Eingabe endet bei Auslösen einer beliebigen Funktionstaste sowie bei Überschreitung der maximalen Eingabelänge von 31 Tastendrücken.
Nach jedem Reset wird die Funktion der Taste <F8> ausgeführt. Es ist also beispielsweise möglich, automatisch ein weiteres Programm in den Computer zu laden und zu starten, indem die entsprechenden Befehle für <F8> eingegeben werden. Sie können auf diese Weise Paradoxon Basic auch direkt mit eigenen Programmen verknüpfen. Nun zu den verbesserten Basic-Befehlen:

▶ `GOTO X`

So ist es möglich, nach GOTO und GOSUB auch einen beliebigen numerischen Ausdruck (Variable oder Formel) anzugeben. Beispiel: GOTO (A*2)/B
An der daraus berechneten Zeilennummer wird das Programm fortgesetzt. Um bei einem

RENUMBER (Umnumerieren) jedoch Fehler zu vermeiden, sollte der zu berechnende Ausdruck nicht mit einer Zahl beginnen. Falsch wäre beispielsweise GOTO 2*A, richtig hingegen GOTO A*2.

▶ `RESTORE X`

Der Zeiger auf das nächste DATA-Element läßt sich auf jede Zeilennummer (X) einstellen, die über einen numerischen Ausdruck eingegeben wird. Existiert die Zeilennummer nicht, werden die DATAs von der nächstmöglichen Position gelesen.

▶ `PRINT (X,Y)"TEXT"`

Nach dem PRINT-Befehl kann man in Klammern die Bildschirmposition angeben, an welcher der Ausdruck stattfinden soll. Die linke obere Ecke des Bildschirms ist dabei die Position (0,0). Die Maximalwerte für die Cursorposition sind also 39 und 24. Achtung: Ein numerischer Ausdruck nach PRINT darf nicht mit »(« beginnen, weil er sonst für die Koordinatenangabe gehalten wird.

▶ `DIM X(N)`

Man kann unter Paradoxon-Basic ein Array beliebig oft neu dimensionieren. Das alte Array wird dabei gelöscht und das neue generiert. Wird ein Array nicht mehr gebraucht, so wird es mit (0) dimensioniert; es belegt dann fast keinen Speicherplatz mehr.

▶ `DEFF1="..." bis DEFF8="..."`

Die Funktionstasten lassen sich auch vom Programm aus definieren, wobei ein beliebiger String-Ausdruck Verwendung findet. Es lassen sich sogar mehr als 31 Zeichen pro Taste speichern, wenn man dafür eine andere Taste unbelegt läßt.

Diskettenbefehle: <@>

Diskettenbefehle können sowohl im Direkt- als auch im Programm-Modus verwendet werden. Sie werden alle mit dem Symbol <@> und der Gerätenummer eingeleitet. Wird keine Nummer angegeben, so bezieht sich der Befehl auf die Geräteadresse 8. Es ist auf jeden Fall zu beachten, daß kein File mit der logischen Filenummer 15 in offenem Zustand befindlich ist.

<@> Der Fehlerkanal wird ausgelesen und auf dem Bildschirm angezeigt.

<@>$ Das Inhaltsverzeichnis einer Diskette wird auf dem Bildschirm ausgegeben. Dabei kann die Ausgabe durch <RUN/STOP> unterbrochen oder durch Drücken einer anderen Taste angehalten (und fortgesetzt) werden.

Soll ein Diskettenbefehl zur Floppy gesendet werden, hängt man ihn hinter <@> an.

Abschließend die Toolkit-Funktionen, welche sinnvollerweise nur im Direktmodus zur Verfügung stehen:

▶ `MERGE "file"`

Ein Programm wird hinter das bereits im Speicher stehende Programm geladen und mit diesem zu einem einzigen größeren Programm verbunden. Das im Speicher stehende Programm muß kleinere Zeilennummern als das nachgeladene haben, weil sonst kein lauffähiges Programm entsteht. Für die Geräteadresse gilt das bei LOAD Gesagte.

▶ `DELETE anfangszeile -- endzeile`

Der angegebene Bereich eines im Speicher stehenden Programms wird gelöscht, bei Löschen eines ganzen Programms sollte man jedoch NEW vorziehen.

▶ `DELETE -70` löscht vom Anfang bis Zeile 70
▶ `DELETE 80-` löscht von Zeile 80 bis Ende

▶ `OLD`

Ein durch NEW oder Reset gelöschtes Programm wird augenblicklich wiederhergestellt. Vorsicht: Ein DELETE-Befehl wird durch OLD nicht widerrufen, da dieser Speicherbereiche umkopiert und somit unwiederbringlich löscht.

▶ `RENUMBER startzeile , abstand`

Die Zeilennummern eines Programms werden entsprechend den Angaben neu numeriert. Dabei werden auch alle Adressen hinter GOTO, GOSUB, THEN, RESTORE, ON GOTO und ON GOSUB aktualisiert. Werden keine Angaben »startzeile,abstand« gemacht, nimmt Paradoxon-Basic den Befehl RENUMBER 100,10 an.

▶ `AUTO startzeile , abstand`

Automatische Zeilennumerierung bei der Programmeingabe erleichtert es, sich an ein fest vorgegebenes Numerierungsschema zu halten. Werden hinter AUTO keine weiteren Werte angegeben, wird mit Zeile 100 begonnen und in Zehnerschritten weitergezählt.

▶ `VAR`

Dieser Befehl gibt alle bisherigen Variablen in der Reihenfolge ihrer Definition auf dem Bildschirm aus. Die Ausgabe kann mit <RUN/STOP> unterbrochen bzw. durch Drücken einer beliebigen Taste angehalten werden. OPEN1,4:CMD1:VAR leitet die Ausgabe auf den Drucker um, was das Austesten eines Programms sehr erleichtert.

▶ `ARRAY`

listet sämtliche Arrays mit ihren Inhalten auf. Arrays mit mehr als sieben Dimensionen werden nicht ausgegeben. Falls man solche Arrays benötigt, sollten diese ganz am Ende definiert werden, da sonst die folgenden Arrays ebenfalls nicht ausgegeben werden.

▶ `FIND ausdruck`

sucht einen beliebigen Ausdruck in einem Programm und gibt die entsprechenden Programmzeilen aus, in denen dieser Ausdruck auftaucht. Bei der Suche eines Strings ist es zweckmäßig, auf das Anführungszeichen am Ende zu verzichten. Zu suchende Basic-Befehle dürfen nicht abgekürzt werden.

5.4.3 Änderung der Standardwerte

Die Anpassung von Paradoxon Basic an individuelle Bedürfnisse erfordert nicht einmal die Kenntnis des Quelltextes. Einige POKE-Befehle genügen, und schon haben Sie Ihr »persönliches« Entwicklungssystem.

1. Allgemeine Änderungen

- Paradoxon Basic laden, aber nicht starten.
- POKE 5333, Rahmenfarbe
  POKE 5338, Hintergrundfarbe
  POKE 5343, Zeichenfarbe
  POKE 2755, Geräteadresse für LOAD u.v.a.
  POKE 2758, Sekundäradresse für LOAD u.v.a.
  POKE 3033, Geräteadresse für Diskettenbefehle
  POKE 5297, Nummer der bei Reset ausgeführten F-Taste
  POKE 5301, Wert dieser Funktionstaste für:
  F1 = 0 , F3 = 32 , F5 = 64, F7 = 96, F2 = 128, F4 = 160, F6 = 192, F8 = 224
- Paradoxon Basic speichern (unter neuem Namen)

2. Funktionstastenbelegung vordefinieren

- Paradoxon Basic laden und diesmal starten
- Funktionstasten nach Belieben programmieren
- Paradoxon-Basic erneut laden, aber nicht starten
- Gegebenenfalls die allgemeinen Änderungen von 1. durchführen
- FOR I = 0 TO 255:POKE 4616+I,PEEK(64512+I):NEXT I im Direktmodus ausführen
- Paradoxon Basic speichern (unter neuem Namen)

5.5 Übersichtliches Listing

Die kleine Erweiterung »UEBERS.LISTING« spaltet beim Listen eine Programmzeile in mehrere Zeilen auf. Dadurch wird die Analyse auch komplizierter Programmteile zum Kinderspiel. Die Idee zu dieser Erweiterung kam dem Programmautor Frank Barcikowski durch den Einzeiler-Wettbewerb im 64'er-Magazin, bei dem die abgedruckten Programme durch ihre gedrängte Darstellung schlecht lesbar waren. Damit ist nun Schluß. Wenn in einer Programmzeile ein Doppelpunkt vorkommt, so wird der darauffolgende Befehl in die nächste Zeile geschrieben. Beispiele kennen Sie aus den zahlreichen Listings in diesem Buch, die oft zusätzlich, oft auch ausschließlich in diesem Format ausgegeben wurden.
Die Erweiterung wird über LOAD"UEBERS.LISTING",8 geladen, mit RUN gestartet und belegt keinen Basic-Speicherplatz. Sie läßt sich aber beliebig verschieben, damit sie auch bei Basic-Erweiterungen funktioniert, die den Bereich ab 49152 benutzen (dazu muß nur der Wert der Variablen ADRESSE in Zeile 0 geändert werden). Die 87 Byte lange Erweiterung läßt sich mit POKE 2,0 einschalten und mit POKE 2,1 wieder ausschalten.
Die Routine »Übersichtliches Listing« ersetzt kurz gesagt jeden Doppelpunkt bei der LIST-Ausgabe durch einen Sprung in die nächste Zeile, wobei die Linksbündigkeit garantiert ist (kein Befehl erscheint auf gleicher Höhe wie die Zeilennummer).
Sie ist dabei in erster Linie für das Arbeiten mit einem Drucker gedacht, da der Vorteil der erhöhten Übersichtlichkeit bei einem Befehl pro Zeile am Bildschirm schnell zu einem großen Nachteil würde, zumindest bei längeren Programmen. »Übersichtliches Listing« dient also zur Analyse fremder, unübersichtlich geschriebener Programme unter Zuhilfenahme eines Drucker-Listings.

5.6 Cross-Ref 64 – Basic-Programme unter der Lupe

Wenn man zu einem Basic-Programm eine Liste aller Sprungadressen, Schleifen und Variablen hat, ist das eine unermeßliche Hilfe, nicht nur während des Programmierens, sondern auch zur nachträglichen Dokumentation. Eine solche Liste erstellt das Maschinenprogramm »CROSS-REF 64«.
Es durchsucht in zwei Durchgängen (Passes) ein Basic-Programm nach Sprungbefehlen und Variablen. Sämtliche Sprungziele und Variablennamen werden dann auf dem Drucker in Form einer Tabelle ausgegeben. Eine solche Tabelle nennt man Cross-Reference. Das Basic-Programm kann dazu übrigens irgendwo im Speicher liegen; es ist an Adresse $0801 nicht gebunden. Dies ist wichtig, wenn der untere Speicherteil für Sprites reserviert wird.

1. Durchgang: Im ersten Durchlauf (Pass) werden die Sprunganweisungen und FOR-NEXT-Schleifen durchsucht und alle Nummern der Zeilen ausgegeben, die Sprunganweisungen enthalten oder ihrerseits angesprungen werden. Dabei werden

alle Sprünge des Basic 2.0 berücksichtigt und FOR-NEXT-Schleifen durch die Laufvariable bezeichnet. Eventuell vorhandene, noch nicht zu Ende geschriebene FOR-NEXT-Schleifen werden mit einem »*« markiert.

2. Durchgang: Hier werden alle Basic-Variablen gesucht und ausgegeben.

Die Ausgabe der Cross-Reference erfolgt entweder auf Bildschirm oder Drucker. Bei Bildschirmausgabe ist das Druckerformat von 80 Zeichen/Zeile unter Umständen störend, aber die Dokumentation auf dem Bildschirm ist wohl auch wenig sinnvoll. Das Programm ist an einen Epson-Drucker mit Görlitz-Interface angepaßt. Es kann aber mit Leichtigkeit so verändert werden, daß es auch für andere Drucker geeignet ist. Es müssen nur vier Speicherzellen mit POKEs geändert werden:

```
POKE 50519,geräteadresse:POKE 50529,geräteadresse
POKE 50523,sekundäradresse+96:POKE 50539,sekundäradresse+96
```

Das Programm belegt den Speicherbereich $C000 (#49152) bis $CAB6 (#51894). Durch folgende Eingabe (siehe Einzeiler #7) speichern Sie eine geänderte Version:

```
SYS57182"NAME",8:POKE193,0:POKE194,192:POKE174,182:POKE175,202:SYS62957
```

Nun noch einige Hinweise zur Anwendung:

- Das Programm wird mit LOAD"CROSS-REF 64",8,1 geladen, woraufhin man NEW eingeben sollte. Später wird es mit »SYS49152,Parameterliste,Bereich« gestartet. Die Parameterliste besteht aus genau fünf Zeichen (Leerzeichen nicht mitgerechnet!). Folgende Zeichen sind erlaubt:

  1. Stelle: P – Ausgabe auf Drucker
  2. Stelle: S – Ausgabe der Zeilen mit Sprungbefehlen
  3. Stelle: J – Ausgabe der angesprungenen Zeilen
  4. Stelle: V – Ausgabe der benutzten Variablen in Reihenfolge des Auftretens
  5. Stelle: J – Ausgabe der benutzten Variablen in sortierter Reihenfolge (1. Typ: real, integer, string, array; 2. Name; 3. Zeilennr.)

  Werden andere Zeichen, z.B. ».«, an der entsprechenden Stelle angegeben, so teilt man »Cross-Ref« damit mit, den entsprechenden Programmpunkt nicht auszuführen.
  Die Bereichsangabe ist optional und hat die gleiche Syntax wie LIST. Nun ein Beispiel:

  ```
  SYS 49152,PS..J,100-200
  ```

  Es werden nun alle Sprungbefehle und Variablen im Zeilenbereich 100–200 untersucht, die Ausgabe erfolgt auf den Drucker.

- Sollten offene FOR-NEXT-Schleifen (NEXT fehlt) vorhanden sein, wird nach einem »*« die Anzahl der fehlenden NEXT-Befehle angezeigt. Die Zahl »255« steht für ein überschüssiges NEXT (NEXT without FOR).

- Bei mehreren Sprüngen in einer Basic-Zeile wird die Zeilennummer nur einmal angegeben.
- Bei ON X GOSUB wird X durch »–« ersetzt, da auch komplizierte Formeln für »X« zulässig sind.
- Bei mehreren Sprüngen hinter einem THEN wird jeweils ein THEN vor das Sprungziel gestellt. Beispiel für eine solche Befehlsfolge: IF X=1 THEN GOSUB 100:GOSUB 200:GOTO 300
- Bei Zuordnung eines neuen Wertes für eine Variable wird die Zeilennummer mit einem »*« gekennzeichnet.
- Die Ausgabe kann jederzeit durch Tastendruck gestoppt werden, eine begonnene Druckzeile wird allerdings zu Ende gedruckt. Mit <RUN/STOP> kann die Ausgabe unterbrochen werden.

5.7 Basic-Programme kürzen

Dieses Programm »BASIC-PACKER« verkürzt jedes Basic-Programm, indem nicht benötigte Leerzeichen, REM-Zeilen usw. aus dem Programm entfernt werden. Zusätzlich können optional die Basic-Zeilen, die Editor-bedingt nur 88 Zeichen lang sein dürfen, auf bis zu 255 Zeichen pro Zeile zusammengeschoben werden. Dies ist gelegentlich wichtig, wenn man für Datenbereiche Platz benötigt, um den OUT OF MEMORY ERROR zu vermeiden.
Danach ist zwar das Programm nich mehr editierbar und auch die Lesbarkeit des Programms hat etwas gelitten, aber es läuft sogar etwas schneller als das Original, weil der Basic-Interpreter weniger Programmcode durchforsten muß; und außerdem hat man Speicherplatz gewonnen. Und das ist bei vielen Basic-Programmen ja auch der Effekt, den man erreichen will.
Der Basic-Packer wird mit LOAD "BASIC-PACKER",8 geladen und durch RUN gestartet. Er verschiebt sich selbständig in den $C000-Bereich, wo er später durch SYS49152 gestartet wird. Nach Abschluß des Packens erfahren Sie noch, wie viele Bytes eingespart wurden. Das »gepackte« Programm ist jetzt zum Speichern bereit.

5.8 Das Maß der Dinge

Das exakte Ausmessen eines Unterprogrammes ist eine große Hilfe, um Laufzeiten gezielt zu verringern. Meist steht jedoch keine hinreichend genaue Uhr zur Verfügung: Um Laufzeiten von Programmen messen zu können, reicht die Stoppuhr für den 100-Meter-Lauf bei weitem nicht mehr aus. Vor allem bei Programmen, bei welchen die menschliche Reaktionsgeschwindigkeit weit hinter der geforderten Genauigkeit zurückbleibt, darf das Meßergebnis nicht von der Fingerfertigkeit des Stoppers abhängen. Doch auch die eingebaute interruptgesteuerte Uhr TI$ des C64 versagt, wenn es um das Auszählen sehr kurzer Programme geht. Ladevorgänge

können damit überhaupt nicht erfaßt werden, weil TI$ während des Ladens »stillsteht«. Abhilfe schaffen hier nur die Timer der CIAs.
Das LMS (Laufzeit-Meß-System) wurde von Franz Stoiber entwickelt, um die Laufzeit von Basic-Programmen zeilenbezogen messen zu können. Weiterhin sollte die Anzahl einzelner Aufrufe von Programmzeilen festgestellt werden, um deren Wichtigkeit innerhalb eines Programms einzuordnen.

Einsatzmöglichkeiten

Das LMS ist im wesentlichen als Werkzeug für folgende Anwendungsfälle konzipiert:

1. Performance-Verbesserung von Basic-Programmen

Durch den Einsatz des LMS ist sofort erkennbar, wo die zeitmäßig kritischen Stellen eines Programms sind. Oft kann durch einige Veränderungen von Basic-Statements wertvolle Laufzeit eingespart werden. Sollte das nicht ausreichen, so ist der Einsatz von Maschinenroutinen angebracht. Die Erfahrung zeigt, daß es selten notwendig ist, ein Programm vollständig in Assembler zu schreiben. Meist bringt ein begrenzter Einsatz von Maschinenroutinen an zeitkritischen Stellen ausreichende Geschwindigkeit unter Beibehaltung der Vorteile von Basic. Diese sind vor allem die wesentlich bessere Lesbarkeit und Wartung sowie der geringere Erstellungsaufwand.

2. Testhilfe

Das LMS stellt die Anzahl der Aufrufe von Basic-Zeilen fest. Es ist damit leicht überprüfbar, ob Programmzeilen überhaupt durchlaufen werden und, falls ja, ob die Anzahl der Aufrufe stimmt. Mit dem LMS können auch Statements gemessen werden, welche mit TI$ – einmal ganz abgesehen von der geringen Genauigkeit – nicht zu erfassen sind (z.B. INPUT#).

Bedienung des LMS

Die Dateien »LMS«, »LMS.PHA« und »LMS.BAS« bilden zusammen das gesamte LMS. Nach Laden und Starten der Datei »LMS« erscheint nach kurzer Zeit ein Menü mit den folgenden Wahlmöglichkeiten:

Messen (m)

Durch die Auswahl der Funktion »Messen« wird das eigentliche Meßprogramm aktiviert. Dies ist immer an der roten Rahmenfarbe erkennbar. Es wird eine Einschaltmeldung geschrieben und danach in den READY-Modus verzweigt. Jetzt muß das zu messende Programm eingegeben oder geladen werden. Bei jedem Start des Programms werden für jede Programmzeile Laufzeit und Aufrufanzahl gespeichert. Ein Start mit GOTO oder CONT ist möglich, führt aber nicht zur Messung. Durch die Eingabe von END im Direktmodus (also nach READY) wird das Meßprogramm ausgeschaltet, und es verzweigt anschließend wieder zum Hauptmenü, in dem die Meßergebnisse des letzten Programmlaufs ausgewertet werden können.

Anzeige ab Nr. (a)

Diese Funktion gestattet die Eingabe einer Zeilennnummer, ab welcher die Meßergebnisse am Bildschirm angezeigt werden. Der Bildschirm zeigt die jeweilige Zeilennummer, die Anzahl der Aufrufe in dieser Zeile und die Gesamtverweilzeit des Basic-Interpreters in dieser Zeile. Bei einer Aufrufanzahl > 1 wird zusätzlich die durchschnittliche Laufzeit pro Aufruf angezeigt; die jeweils größte Laufzeit ist durch < ← > zur besseren Orientierung markiert.

Vorhergehende Seite (< ↑ >)

Durch Drücken der Taste < ↑ > wird die Ausgabe am Bildschirm um eine Seite zurückgeblättert.

Nächste Seite (RETURN)

Dies blättert die Bildschirmausgabe um eine Seite nach vorne.

Summe (s)

Diese Funktion erlaubt die Berechnung der Programmlaufzeit, bezogen auf einen einzugebenden Zeilenbereich.

Drucken (d)

Die Auswahl dieser Funktion bewirkt die Ausgabe der Meßergebnisse auf dem Drucker. Durch die Angabe von Zeilennummern läßt sich dieser Bereich begrenzen.

Hilfe (h)

Ein Menü wird hier angezeigt.

Ende (e)

Der Computer wird nach Bestätigung der Auswahl in den Reset-Zustand versetzt.

Allgemeine Hinweise zu LMS

Allgemein ist zu beachten, daß bei sofortigem <RETURN> bei Eingabe der Zeilennummer das gesamte Programm durchsucht wird (0–63999). Das Meßverfahren kann auf Programme mit bis zu 1022 Zeilen angewandt werden; bei unnormaler Speicheraufteilung (Paradoxon Basic o.ä.) funktioniert LMS ebensowenig wie bei selbstmodifizierenden Programmen oder Veränderungen des NMI-Timers (RS-232-Programme). Der Basic-Speicher ist auf 30 Kbyte begrenzt; die maximal meßbare Laufzeit pro Zeile beträgt etwa 72 Minuten; pro Zeilenaufruf benötigt das LMS etwa 0.9 Millisekunden, was während der Messung zu einer Verlängerung der Programmlaufzeit führt. Die Meßergebnisse sind jedoch Netto-Laufzeiten, d.h., sie entsprechen den Zeiten, die ohne LMS entstehen. Programmschleifen innerhalb einer Basic-Zeile wie

```
10 GET A$:IF A$="" THEN 10
```

werden nicht erkannt, da LMS zeilenorientiert vorgeht. Um die exakten Aufrufzahlen zu erhalten, sind die Zeilen aufzuteilen:

```
10 GET A$
20 IF A$="" THEN 10
```

Dies gilt auch für Zeilen, in welchen nach GOSUB weitere Statements folgen, da sonst die durchschnittlich ausgewiesene Laufzeit mit Vorsicht zu genießen ist.
Abschließend die Listings der beiden Basic-Teile:

Listing 5.4: LMS (Ladeprogramm)

```
100 rem ----------- laufzeit-mess-system
110 rem --------------------ladeprogramm
120 rem --------------------------------
130 rem -------------- (c) franz stoiber
140 rem ------------- braeunlichgasse 30
150 rem ------------- a-2700 wr.neustadt
160 rem --------------------- oestereich
170 rem --------------------------------
180 if l=1 then 260
190 poke 53280,15: poke 53281,15: poke 646,6
200 ga=8
210 print chr$(14);"(clr)(down)(down)(down)(down)(down)(down)(down)(down)(down)(
down)(down)(rght)(rght)(rght)(rght)(rght)(rght)Laufzeit-Mess-System  V2.0"
220 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(c) Franz Stoiber  11/85"
230 print "(down)(down)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)Bitte warten"
240 poke 646,15
250 l=1: load "lms.pha",ga,1
260 sys 40698
270 print "(home)";
280 print "(down)(down)sys 40780"
290 print "(down)(down)load ";chr$(34);"lms.bas";chr$(34);",";ga
300 print "(down)(down)(down)(down)run"
310 for i=631 to 633: poke i,13: next
320 poke 198,3
330 print "(home)";
340 rem --------------------------------
350 rem fuer das arbeiten mit datassette
360 rem sind folgende zeilen zu aendern:
370 rem
380 rem *** 200 ga=1
390 rem *** 210 entfaellt
400 rem *** 220 entfaellt
410 rem *** 230 entfaellt
420 rem *** 240 poke 646,0
430 rem *** 270 print "(clr)";
440 rem *** 300 print "(down)(down)(down)(down)(down)run"
450 rem
460 rem --------------------------------
470 rem am band muessen die programme
480 rem in der reihenfolge
490 rem
500 rem lms, lms.pha, lms.bas
510 rem
520 rem gespeichert werden.
```

Listing 5.5: LMS.BAS (Auswertungsprogramm)

```
100 rem ------ lms-auswerteprogramm v2.0
101 rem ------- 11/85  (c) franz stoiber
110 print chr$(14);chr$(8): poke 808,225: zf=6
120 e$="—————————————— Ende ——————————————"
130 poke 53280,15: poke 53281,15: poke 646,zf: gosub 1440
140 bl$="                                        "
150 ra=53248: pu=253: l=22: cl=985.248: r=.05
160 k0=2↑8: k1=2↑16: k2=2↑24: k3=2↑32: k4=k1-1: k5=63999: k6=10
170 rd=40679: at=40710: ab=40535: nr=40918
180 p1=40926: p2=p1+1: p3=p1+2: p4=p1+3: p5=p1+4: p6=p1+5: p7=p1+6: p8=p1+7
200 j=0: k=80: p=ra
210 kz$="       Nr   Anz        msek    msek/Anz"
220 sys at,6,5,"Messung ..............   m"
230 sys at,6,7,"Anzeige ab Nr ........   a"
240 sys at,6,9,"Vorhergehende Seite ..   ↑"
250 sys at,6,11,"Naechste Seite ....... return"
260 sys at,6,13,"Summe ................    s"
270 sys at,6,15,"Drucken ..............    d"
280 sys at,6,17,"Hilfe ................    h"
290 sys at,6,19,"Ende .................    e"
300 poke 198,0: sys at,0,24,"(rvs)    Weiter mit: m/a/↑/return/s/d/h/e   (off)";
310 poke 2023,160: poke 56295,zf
320 get a$: if a$="" then 320
330 if a$="m" then 430
340 if a$="a" then 450
350 if a$="↑" then 490
360 if a$=chr$(13) then 540
370 if a$="s" then 570
380 if a$="d" then 720
390 if a$="h" then 1010
400 if a$="e" then 1030
410 goto 300
420 rem ------------------------ messung
430 poke 808,237: print chr$(9);"(clr)";: sys 40160
440 rem --------------------- anzeige ab
450 gosub 1930: if f then 300
460 gosub 1440: gosub 1480
470 p=p-8: gosub 1130: goto 300
480 rem ------------------------ zurueck
490 gosub 1930: if f then 300
500 p=p-(zz+1-1)*8
510 if p<ra then p=ra
520 gosub 1130: goto 300
530 rem ---------------------------- vor
540 gosub 1930: if f then 300
550 gosub 1130: goto 300
560 rem -------------------------- summe
570 gosub 1930: if f then 300
580 gosub 1440
590 sys at,5,4,"Summe": sys at,5,5,"—————"
600 gosub 1480: gosub 1580
610 p=p-8: s=0
620 gosub 1080
630 if z>zb then 670
640 gosub 1370
650 s=s+t: sys at,30,10,z
660 goto 620
670 sys at,30,10,"       "
680 t=s: t$="": gosub 1400
690 sys at,5,17,"Summe =";t$;" msek"
700 p=ra: goto 300
710 rem ------------------------ drucken
720 gosub 1930: if f then 300
730 gosub 1440
```

```
740 sys at,5,4,"Drucken": sys at,5,5,"────────"
750 gosub 1480: gosub 1580: p=p-8: s=Ø
760 sys at,5,15,"Drucker O.K. ?"
770 sys at,5,16,"Weiter mit beliebiger Taste"
780 poke 198,Ø: wait 198,1: poke 198,Ø
790 open 4,4,7: poke 768,185: print#4,"";: poke 768,139
800 if st then sys at,8,22,"(rvs) Drucker nicht bereit (off)": close 4: goto 300
810 if k<80 then pl$="": gosub 1830: goto 830
820 sys at,Ø,15,bl$;: sys at,Ø,16,bl$;
830 f=Ø: pl$=""
840 gosub 1080
850 if z>zb then 960
860 gosub 1320
870 gosub 1370
880 s=s+t: t$="": b$="                    "
890 gosub 1400
900 if az<2 then 920
910 b$="": t=t/az: az=Ø: goto 890
920 pl$=pl$+"  "+z$+az$+t$+b$
930 if f=Ø then f=1: goto 840
940 gosub 1830
950 goto 830
960 if f then gosub 1830
970 t=s: t$="": gosub 1400
980 pl$="          Summe ="+t$+" msek": gosub 1830
990 close 4: p=ra: goto 300
1000 rem ------------------------- hilfe
1010 gosub 1440: goto 220
1020 rem -------------------------- ende
1030 gosub 1440: sys at,5,13,"Bitte Ende bestaetigen (j/n): (rvs) (off)";
1040 get a$: if a$="" then 1040
1050 if a$="j" then sys at,35,13,"j": sys 64738
1060 sys at,Ø,13,bl$;: goto 220
1070 rem ---------- werte aus ram lesen
1080 p=p+8: ph=int(p/kØ): pl=p-ph*kØ
1090 poke pu,pl: poke pu+1,ph: sys rd
1100 z=peek(p1)+peek(p2)*kØ
1110 return
1120 rem ----------- ausgabe bildschirm
1130 print "(clr)(rvs)    Nr   Anz        msek     msek/Anz    "
1140 m1=Ø: m2=Ø: c1$=" ": c2$=" "
1150 for zz=1 to l
1160 :gosub 1080
1170 :if z=k4 then p=p-8: print e$;: goto 1280
1180 :gosub 1320
1190 :gosub 1370
1200 :if t>m1 then m1=t: l1=zz: c1$="←"
1210 :t$=""
1220 :gosub 1400
1230 :if az<2 then 1260
1240 :t=t/az: if t>m2 then m2=t: l2=zz: c2$="←"
1250 :az=Ø: goto 1220
1260 :print z$;az$;t$
1270 next
1280 if c1$="←" then sys at,24,l1+1,c1$;
1290 if c2$="←" then sys at,36,l2+1,c2$;
1300 return
1310 rem ---------- aufbereitung nr,anz
1320 z$=right$("      "+str$(z),6)
1330 az=peek(p3)+peek(p4)*kØ
1340 az$=right$("      "+str$(az),6)
1350 return
1360 rem -------------- zeit errechnen
1370 t=(peek(p5)+peek(p6)*kØ+peek(p7)*k1+peek(p8)*k2)/cl
1380 return
1390 rem ------------ aufbereitung zeit
1400 tr=t+r: th=int(tr): tl=int((tr-th)*k6)
```

```
1410 t$=t$+right$("          "+str$(th),10)+"."+right$(str$(tl),1)
1420 return
1430 rem ------------------- kopfzeilen
1440 print "(clr)(rvs)  Laufzeit  - Mess  -  System   V2.0   ";
1450 print "(rvs)  (c) 11/85  Franz Stoiber                 "
1460 return
1470 rem ---------- eingabe zeilennr ab
1480 ro=8: sys at,5,ro,"Ab Nr  ?  ";: gosub 1650
1490 if a$="     " then za=0: sys at,14,ro,za: goto 1520
1500 za=val(a$): if za<=k5 then sys at,0,22,bl$;: goto 1520
1510 sys at,6,22,"(rvs)    Zeilennummer zu gross    (off)";: goto 1480
1520 zh=int(za/k0): zl=za-zh*k0: poke nr,zl: poke nr+1,zh
1530 sys ab: p=peek(pu)+peek(pu+1)*k0
1540 p=p-8: gosub 1080
1550 if z<k4 then return
1560 sys at,6,22,"(rvs) Zeilennummer nicht gefunden (off)";: goto 1480
1570 rem --------- eingabe zeilennr bis
1580 ro=10: sys at,5,ro,"Bis Nr ?  ";: gosub 1650
1590 if a$="     " then zb=k5: sys at,14,ro,zb: return
1600 zb=val(a$): if zb>=za then 1620
1610 sys at,5,22,"(rvs)  Zeilennr bis < Zeilennr ab  (off)";: goto 1580
1620 if zb>k5 then zb=k5
1630 sys at,0,22,bl$;: return
1640 rem ---------------- zahl eingeben
1650 n=1: rc=1024+14+40*ro: co=rc+54272: poke 198,0
1660 for i=1 to 5: poke co+i,zf: next
1670 if n>5 then n=5
1680 if n<1 then n=1
1690 po=rc+n: poke po,peek(po)+128
1700 get c$: if c$="" then 1700
1710 if c$>="0" and c$<="9" then poke po,asc(c$): n=n+1: goto 1670
1720 if c$=chr$(157) then poke po,peek(po)-128: n=n-1: goto 1670
1730 if c$=chr$(29) then poke po,peek(po)-128: n=n+1: goto 1670
1740 if c$=chr$(20) then sys at,15,ro,"     ";: goto 1650
1750 if c$=chr$(13) then poke po,peek(po)-128: goto 1770
1760 goto 1700
1770 a$=""
1780 for i=1 to 5
1790 :a$=a$+chr$(peek(rc+i))
1800 next
1810 return
1820 rem ---------------- zeile drucken
1830 if k<68 then 1900
1840 if k<73 then print# 4,"": k=k+1: goto 1840
1850 j=j+1: print# 4,"": print# 4,""
1860 print# 4,"   Laufzeit - Mess - System  V2.0";
1870 print# 4,"      (c) Franz Stoiber 11/85      Seite: ";right$(str$(j+100),2)
1880 print# 4,"": print# 4,"": print# 4,kz$;kz$: print# 4,""
1890 k=8
1900 print# 4,pl$: k=k+1
1910 return
1920 rem -------------- test auf messung
1930 pp=p: f=0: p=ra-8: a$=""
1940 gosub 1080
1950 for i=p1 to p8: a$=a$+chr$(peek(i)): next
1960 if a$="lms-f.s." then 1990
1970 sys at,6,22,"(rvs) Keine Messung gelaufen ! (off)"
1980 f=1: return
1990 gosub 1080
2000 if z<k4 then p=pp: return
2010 sys at,6,21,"(rvs) Waehrend der Messung war "
2020 sys at,6,22,"(rvs)  kein Programm geladen!  (off)"
2030 goto 1980
```

6 Effektives Programmieren in Basic oder: Schneller, schneller, schneller

Für den C64 ist Basic die »unkomplizierteste« Programmiersprache: Basic 2.0 ist fest eingebaut, hat keinen unüberschaubar großen Befehlssatz und läßt sich auch für Einsteiger erlernen. So weit, so gut. Doch die Geschwindigkeit der Ergebnisse, die man normalerweise mit Basic erzielt, läßt auch schon bei scheinbar leichten Aufgaben bald zu wünschen übrig. Deshalb widmet sich dieses Kapitel ganz besonders dem Thema »Wie poliere ich ein fertiges Basic-Programm auf, damit es mehr Leistung bringt?«, kurz als »Performance-Verbesserung« bezeichnet. Teilweise sind diese Hinweise bereits in den Anfangsstadien des Programmentwurfes zu beachten, teilweise für die nachträgliche Optimierung vorgesehen.
Über eines muß man sich dabei jedoch im klaren sein: Will man ein Programm möglichst schnell ablaufen lassen, so erzielt man damit nicht immer (aber oft!) die speichersparsamste Lösung, und die Lesbarkeit läßt deutlich nach.
Noch ein Hinweis vorweg: Das Laufzeit-Meßsystem (»Das Maß der Dinge«) aus dem letzten Kapitel ist eine riesige Hilfe bei der Programmoptimierung.

6.1 Der Basic-Interpreter

Wie gerade erwähnt, verfügt der C64 im Gegensatz zu vielen anderen Computern über einen fest integrierten Basic-Interpreter (siehe »C64 für Insider«). Diesen wollen wir nun unter die Lupe nehmen, weil dies das Verständnis für Basic schärft.
Zunächst muß man wissen, daß der C64 »von Natur aus« kein Wort Basic spricht. Seine Bausteine verarbeiten lediglich elektronische Impulse, welche weniger mit Basic zu tun haben als vielmehr mit elektrischer Spannung. Dabei wird jeweils zwischen hohem und niedrigem Spannungspegel differenziert, so daß man – vereinfacht gesagt – nur zwischen Nullen und Einsen, also zwei Zuständen, unterscheiden kann. Nun überspringen wir alle komplizierten Zwischenstufen und halten nur fest, daß aus der Kombination von 0 und 1 schließlich die »Maschinensprache« entsteht. Maschinensprache ist also der direkteste Zugang zum Computer, den man softwaremäßig erreicht; da der C64 also »nur« Maschinensprache versteht, ist auch sein Betriebssystem – das zentrale Steuerprogramm für Ein-/Ausgabe – darin geschrieben.
Wie kommt der C64 nun dazu, Basic zu verstehen? Nun, er »versteht« im Grunde nicht Basic, sondern hat nur ein eingebautes Maschinenprogramm, den Basic-Interpreter, welcher die Eingabe und Abarbeitung von Basic-Programmen übernimmt. Findet er beispielsweise den Befehl

PRINT, so weiß der Interpreter, daß er alles das ausgeben muß, was hinter PRINT steht; ebenso können dabei Fehler auftreten, die er mit entsprechenden Meldungen quittiert.
Der Prozessor selbst vermag jedoch die Basic-Programme nicht von anderen Textdateien zu unterscheiden; er kennt nur »seine« Maschinenbefehle, die er mit maximaler Geschwindigkeit abarbeitet. Nun sind diese Maschinenbefehle recht primitiv im Vergleich zu einfachsten Basic-Statements, so daß der Interpreter zur Ausführung einzelner Basic-Befehle unvorstellbar viele Kommandos in Maschinensprache benötigt.
Und nun sind wir schon beim springenden Punkt: Während ein Maschinenbefehl in 1, 2 oder 3 Mikrosekunden abgearbeitet ist, ist ein Basic-Befehl das »Produkt« ungezählter Maschinenbefehle.
In diesem Kapitel wollen wir nun dem Interpreter seine Arbeit erleichtern, wofür er uns mit kürzeren Ausführungszeiten belohnt. Wenn Sie jedoch dieser Exkurs in die innersten C64-Geheimnisse auf den Geschmack gebracht hat, so rate ich Ihnen, sich »just for fun« ein wenig mit Maschinensprache zu beschäftigen, so daß Sie sich diesen interessanten Themenbereich (siehe »C64 für Insider«) erschließen können.

6.2 Faustregeln zur Beschleunigung

Regel 1

- Häufig vorkommende Zahlen, Adressen und Variable, besonders innerhalb von Schleifen, werden am Anfang des Programms vordefiniert. Dadurch wird schneller auf diese zugegriffen. Ungünstig wäre also folgendes Programm, welches die obersten 11 Bildschirmzeilen mit »A« beschreibt:

```
10 PRINT CHR$(147)
20 Z=0
30 POKE 1024+Z,1:POKE 55296+Z,14
40 Z=Z+1
50 IF Z=374 THEN END
60 GOTO 30
```

Schon mit bloßem Auge sieht man die Verbesserung durch folgende Variablendefinitionen:

```
10 PRINT CHR$(147)
20 Z=0:S=374:V=1024:F=55296:B=1:D=14
30 POKE V+Z,B:POKE F+Z,D
40 Z=Z+1
50 IF Z=S THEN END
60 GOTO 30
```

Damit wurde eine Laufzeitoptimierung von etwa 50 Prozent erreicht!

- Variable, die sich im Lauf des Programms verändern (wie Z im Beispielprogramm), werden trotzdem vordefiniert, allerdings mit einem unschädlichen Ausgangswert (»Dummy«, im Beispiel Z = 0).
- Die Zahl, die am häufigsten vorkommt, wird als erste vordefiniert, damit sie der Basic-Interpreter schneller findet. Im Beispiel gilt dies für Z; die anderen Variablen treten dort etwa mit gleicher Häufigkeit auf.
- Variablennamen sollten möglichst einstellig, höchstens aber zweistellig sein. Im Beispielprogramm wurden zwar einstellige Variablennamen verwendet, doch durch den Einsatz zweistelliger Namen entsteht ein kaum noch meßbarer Zeitverlust. »Halbe Sätze« als Variablennamen hingegen sind wenig effektiv.

Regel 2

- In Schleifen mit aneinandergereihten Druckanweisungen ist PRINT viel schneller als POKE. Im Beispiel wäre dafür folgende Änderung erforderlich:

```
10 PRINT CHR$(147)
20 Z=0:S=374
30 PRINT CHR$(65);
40 Z=Z+1
50 IF Z=S THEN END
60 GOTO 30
```

Damit gewinnen wir gegenüber der letzten Version über 25 Prozent.

- Die PRINT-Variablen sollten entweder vordefiniert oder im Quote Mode eingesetzt werden. Folgende zwei weitere Versionen entstehen dadurch:

1. CHR$(65) vordefiniert als A$

```
10 PRINT CHR$(147)
20 Z=0:S=374:A$=CHR$(65)
30 PRINT A$;
40 Z=Z+1
50 IF Z=S THEN END
60 GOTO 30
```

2. CHR$(65) ersetzt durch »A«

```
10 PRINT CHR$(147)
20 Z=0:S=374
30 PRINT "A";
40 Z=Z+1
50 IF Z=S THEN END
60 GOTO 30
```

Zwischen diesen beiden Versionen besteht kein erkennbarer Laufzeitunterschied, sieht man von der Anzahl der Zeichen im Programm ab.

Regel 3

- Prüfung auf Ungleichheit (< >) bietet Vorteile, wenn sie eine Zeile erspart.
- Bei Schleifen mit Sprunganweisungen ist IF..GOTO um ein Minimum schneller als IF..THEN. Beispiel:

```
10 PRINT CHR$(147)
20 Z=0:S=374
30 PRINT "A";
40 Z=Z+1
50 IF Z<>S GOTO 30
```

Dadurch sparen wir also eine Zeile und gewinnen eine geringe Zeitspanne.

Regel 4

- Bei IF-THEN-Prüfungen mit mehreren Bedingungen sollen diese Bedingungen in einzelne IF-THEN-Prüfungen hintereinander gesetzt werden. Dabei wird die Bedingung an die erste Stelle gesetzt, welche am seltensten erfüllt ist. Nehmen wir ein theoretisches Beispiel ohne Bezug zu den bisherigen Programmen:

```
100 IF (A=1 AND B=2 AND C=3) THEN 999
110 GOTO 50
```

Zeile 100 prüft jedesmal, ob alle drei Bedingungen erfüllt sind; erst dann wird entschieden, ob das Programm auf Zeile 999 springt oder auf 110 weiterläuft. Nehmen wir an, A ist im 20. Durchlauf erfüllt, B im 50., C aber erst im 300. Durchlauf. Das Programm muß also 300mal alle drei Bedingungen prüfen. Schreiben wir deshalb Zeile 100 so:

```
IF C=3 THEN IF B=2 THEN IF A=1 THEN 999
```

Dann bricht das Programm 300mal die Prüfung nach dem C ab und geht in 110 weiter. B und A werden erst dann zur Prüfung herangezogen, wenn C=3 ist. Es ist wohl klar und einzusehen, daß die zweite Schreibweise schneller ist.

Regel 5

- Schleifen sollen nicht mit IF...THEN, sondern mit FOR...TO...NEXT gebildet werden. Die Schleifenvariable nach NEXT soll dann weggelassen werden, wenn es nicht zu Verwechslungen mit anderen Schleifen führen kann.
  Dazu ändern wir das alte Beispielprogramm (Bildschirmausgabe von 374 mal »A«):

```
10 PRINT CHR$(147)
20 FOR Z=1 TO 374
```

```
30 PRINT "A";
40 NEXT
```

Auf die allererste Programmversion dieses Kapitels bezogen, haben wir jetzt etwa 90 Prozent gewonnen, gegenüber der letzten Fassung (von Regel 3) sind wir fast dreimal so schnell geworden.

Regel 6

- Programme ohne REM-Erläuterungen und ohne Leerstellen zwischen den Zeichen laufen schneller.
- Zur Reduzierung der Zeilenzahl sollte man möglichst viele Befehle in eine Zeile packen. Dies erledigt der »Basic-Packer« aus Kapitel 5 automatisch. Hier können wir sogar »von Hand« packen:

```
10 PRINTCHR$(147);:FORZ=1TO374:PRINT"A";:NEXT
```

Nun haben wir deutlich über 90 Prozent gegenüber dem allerersten Entwurf gewonnen. An diesem Programm gibt es nun auf Basic-Ebene keine Verbesserungsmöglichkeiten mehr.

Regel 7

- Bei einer Multiplikation soll die längere Zahl immer vor der kürzeren stehen (langer Multiplikand, kurzer Multiplikator). Beispiel: 5.127954*3.5 statt 3.5*5.127954
- Eine einzelne Null wird durch einen Punkt ersetzt, eine Null vor dem Dezimalpunkt wird weggelassen. Beispiele:

```
A=.       statt     A=0
A=.5      statt     A=0.5
```

Regel 8

- Die Funktion »Potenzieren« (↑) ist durch Mehrfach-Multiplikation zu ersetzen. Beispiel:

```
A=B*B*B   statt     A=B↑3
```

Dieser simple Trick bringt unglaubliche Ergebnisse!

- Gleichermaßen ist die Multiplikation mit 2 oft auch als Addition einer Zahl mit sich selbst zu schreiben:

```
A=B+B     statt     A=B*2
```

Dies ist jedoch nur für den Faktor 2 zutreffend; ab dem Faktor 3 wird die Multiplikation schneller als die Mehrfach-Addition.

Regel 9

- Der Aufruf von Unterprogrammen mit GOSUB ist schneller als mit GOTO.
- Häufig gebrauchte Unterprogramme gehören ganz an den Anfang eines Programms. Sie müssen dann allerdings erst mit einem GOTO umgangen werden. Ein Beispiel ist etwa:

```
10 GOTO 100
20 ... Unterprogramm ...
...
100 PRINT "HAUPTPROGRAMM"
110 GOSUB 20
```

- Alle Sprünge sind in ihrer Laufzeit auch von der Sprungdistanz abhängig.

Regel 10

- Integervariablen (I%) sind in Basic langsamer als Fließkommavariablen (I); der Speicherbedarf für Integervariablen reduziert sich nur (!) bei Arrays. Bei »kompilierten« Programmen (siehe 6.3) verhält es sich umgekehrt.

Regel 11

- Große Arrays, die am Anfang definiert werden, verlangsamen ärgerlicherweise das Anlegen neuer Variablen.

Regel 12

- Konstanten sollten nicht nur vordefiniert, sondern auch so weit als möglich ausgerechnet werden. Beispiel:

```
A=53281     statt     A=13*4096+33
```

- Das PI-Symbol (SHIFT und ↑) hat eine hohe Rechengenauigkeit, die in vielen Fällen nicht benötigt wird. Eine Definition eines Wertes mit niedrigerer Anzahl von Nachkommastellen bietet sich an:

```
P=3.14              statt     P=<SHIFT ↑>
```

6.3 Basic-Compiler im Kommen

Alle in 6.2 genannten Optimierungsmethoden helfen zwar, ein Basic-Programm zu beschleunigen, doch Basic bleibt eben Basic und ist noch lange nicht die schnellste Programmiersprache. Allerdings ist Maschinensprache schwierig zu erlernen und aufwendig zu programmieren, so daß man meist auf einfaches, aber langsames Basic beschränkt bleibt und die großen Vorteile der Maschinensprache nicht zu nutzen vermag.

6.3.1 Was ist ein Compiler?

Ein echtes Wunderwerk sind in den Augen von Einsteigern daher die sogenannten »Basic-Compiler«. Dies sind Programme, welche Ihre gewöhnlichen Basic-Programme in Maschinenbefehle übertragen. Auf den ersten Blick erscheint dies unmöglich, doch es ist wahr: Ein Compiler nimmt ein Basic-Programm als Grundlage für ein Maschinenprogramm, welches dasselbe bewirkt, aber um ein Vielfaches schneller ist!
Dieser Übersetzungsvorgang unterscheidet sich vom Interpreter dadurch, daß nur einmal – nämlich bei der »Kompilation« – ein lauffähiges Maschinenprogramm entsteht; bei allen späteren Programmläufen fällt der zeitraubende Interpretierungsvorgang weg. Dabei ist zu beachten, daß das »Kompilat« (Ergebnis der Kompilation) die Geschwindigkeit »echter« Maschinenprogramme nicht ganz erreicht. Es wäre zuviel verlangt, wenn ein Compiler für ein Basic-Programm die optimale Assembler-Umsetzung finden sollte. Man spricht deshalb vom sogenannten »Speedcode« (schnell übersetzbarer Programmcode), um Verwechslungen mit »Maschinensprache total« auszuräumen.
Dennoch reicht für viele Anwendungsfälle ein Basic-Compiler völlig aus. Ich habe in meiner intensiven Beschäftigung mit C64-Software schon professionelle Produkte von sensationeller Qualität gesehen, die sich bei allernächster Untersuchung des Programms als Kompilate entpuppten – obwohl selbst reine Maschinenprogramme zu den entsprechenden Themen diesen kompilierten Basic-Programmen nicht das Wasser reichen konnten. Meist stammten diese vom Compiler »Austro-Comp« oder »Austro-Speed«, welcher für C64/128 zu einem langjährigen Standard gereift ist.

6.3.2 Compiler-Anwendung am Beispiel

Wie man mit einem Compiler umgeht, läßt sich am besten im konkreten Fall zeigen. Zunächst schreibt man ein Basic-Programm wie gewohnt, oder wählt ein längst fertiges Programm zur Nachbehandlung mit einem Compiler aus. Dazu benötigt man eine spezielle Arbeitsdiskette, auf welcher man das zu kompilierende Programm ablegt; darauf sollte auch genügend Platz für alle Arbeitsdateien des Compilers sein.
Anschließend lädt man den Compiler; der Austro-Comp fragt jetzt sofort, welche Datei zu kompilieren ist. Man gibt den Namen der Basic-Datei ein, und erhält nach einer kürzeren oder längeren Zeitspanne – der Kompilationszeit – auf der Arbeitsdiskette mindestens noch eine weitere Datei, das Kompilat. Dies ist bei Austro-Comp durch den Vorsatz »C/« gekennzeichnet; es ist bei langen Basic-Programmen kürzer, bei kurzen Basic-Routinen länger als die Basic-Fassung.
Die alte Basic-Datei bleibt natürlich erhalten, denn an einem Kompilat lassen sich keine Basic-Änderungen mehr durchführen; im Falle eines Fehlers oder eines Verbesserungswunsches editiert man also nicht das Kompilat, sondern verändert zuerst den »Quelltext« (Sourcecode), also das eigentliche Basic-Programm und kompiliert dieses erneut.

Sie sehen bereits den Nachteil dieser Methode: Während bei einem Interpreter unmittelbare Änderungen und Testläufe möglich sind, verliert man bei einem Kompilat – welches auch nur über »RUN« zu starten ist – die Interaktivität. Kompilierte Programme lassen sich daher meist nicht über <RUN/STOP> abbrechen.
Auch Eingaben wie A=5:GOTO 210 zum Testen bestimmter Programmsituationen sind bei Kompilaten nicht möglich, da es sich dabei um fertige, nicht mehr zu modifizierende Maschinenprogramme handelt. Dies hat andererseits den Vorteil eines indirekten Schutzes vor unbefugten Eingriffen, weshalb kompilierte Programme oftmals zu Kopierschutz-Zwecken eingesetzt werden.
Da kompilierte Programme aus Maschinencode bestehen, lassen Sie sich entweder überhaupt nicht listen oder ergeben nur eine Zeile wie

```
1985 SYS 2071 AUSTRO-SPEED E1
```

In Kapitel 7 lernen Sie ein solches Kompilat kennen; es befindet sich auf der Programmdiskette unter dem Namen »C/MENUE-BSP. 11« und dürfte Ihnen als Anschauungsbeispiel genügen. Auch der Tool-Creator (Kapitel 5) ist in der auf Diskette beiliegenden Fassung ein Kompilat. An Anschauungsmaterial fehlt es also nicht.

6.3.3 Compiler für andere Programmiersprachen

Obwohl ein Basic-Compiler oftmals eine gelungene Kombination aus einfacher Programmentwicklung und semiprofessionellem Ergebnis garantiert und somit erst »nach« dem Interpreter eingesetzt wird, sind Compiler viel älter als Interpreter.
Manche Programmiersprachen wie Pascal oder C sind beispielsweise fast ausschließlich als Compiler sinnvoll, lediglich zu Lehrzwecken findet man auch Interpreter. Ein Interpreter ist also ausschließlich für die interaktive Programmentwicklung und die Fehlersuche vorteilhaft; die generierten Ergebnisse sprechen eindeutig für Compiler.
Der C 64 stellt Sie nicht vor die Grundsatzfrage »Compiler oder Interpreter«; vielmehr können Sie, ausgehend von der Arbeit mit dem Interpreter, bei Bedarf auch einen Compiler heranziehen.
Eine ganz neue Entwicklung sei auch noch gewürdigt: Die Programmiersprache »TrueBasic«, die sich anschickt, auf größeren Rechnern ein neuer Standard zu werden, bezeichnet sich als »interpretierender Compiler«. Dies ist eine Mischung aus Interpreter und Compiler: Die interaktive Programmentwicklung ist weiterhin möglich, aber der Programmablauf geschieht über eine Art »Speedcode«. Damit wird zwar nicht ganz die Geschwindigkeit reiner Compiler erzielt, aber der große Komfort beim Programmieren ist dadurch gesichert.

6.3.4 Effektives Arbeiten mit Compilern

Alle in 6.2 vorgestellten Tricks, wie man ein Basic-Programm beschleunigt, können Sie nun getrost vergessen, wenn Sie ohnehin vorhaben, ein Programm zu kompilieren. Die erwähnten Regeln erleichtern dem Basic-Interpreter seine Arbeit und erhöhen auch die Kompilationsgeschwindigkeit, doch die Ablaufzeiten verändern sich nicht.
In einer Hinsicht lassen sich jedoch mit Compilern ganz beachtliche Ergebnisse erzielen: Verwendet man viele (oder ausschließlich) Integerberechnungen, so erreicht man Zeiten, die ganz nah am »echten« Maschinencode liegen. Der Zahlenbereich für Integervariablen umfaßt nur die ganzen Zahlen (also ohne Nachkommastellen) im Bereich von –32768 bis +32767; in nichtwissenschaftlichen Basic-Programmen sind oft keine anderen Wertebereiche erforderlich.
Integeroperationen lassen sich nämlich vom Compiler ohne Mühe in effektive Maschinenbefehle übertragen; die Primitivität der Maschinensprache-Anweisungen liegt darin, daß nur mit Integerwerten bequem umzugehen ist. Anweisungen wie »A%=A%+1« oder »A%=A%*2« lassen sich beispielsweise in Assembler als »INC A%« oder »ASL A%« formulieren und sind in Rekordzeiten ausgeführt. Die Berechnung von Fließkommawerten hingegen hält sowohl Interpreter als auch Compiler auf, da hierfür komplizierte Umrechnungen anfallen, die eben bei Verwendung des Integerbereiches einzusparen sind.
Das Kompilat »C/MENUE-BSP. 11« von der Programmdiskette bezieht gerade dadurch einen größten Geschwindigkeitszuwachs, daß es ausschließlich im Integerbereich arbeitet.
Damit nun ein Compiler Integerwerte als solche erkennt und eindeutig von Fließkommazahlen unterscheiden kann, muß man ihm dies durch das Prozentzeichen im Variablennamen (I%, DF%) mitteilen; manche Compiler erlauben auch die nachträgliche »Deklaration« einer Variablen als Integerwert.
Wer sich schon mit den Compilersprachen C oder Pascal beschäftigt hat, kennt diese Unterscheidung nach Datentypen bereits. Es erfordert nur eine gewisse Planung beim Programmieren, um von einer Variablen im voraus zu ersehen, ob sie unbedingt den Fließkomma-Bereich benötigt oder ob die schnelleren Integerrechnungen ausreichen.

6.3.5 Der Ascompiler

Ein etwas ungewöhnlicher Compiler ist nun das Programm »Ascompiler«, seines Zeichens als Listing des Monats in der Ausgabe 1/86 der Zeitschrift »64'er« ausgezeichnet. Dieser übersetzt ein stark vereinfachtes Basic (»Tiny-Basic« genannt) in 100prozentigen Maschinencode und erreicht damit Ausführungszeiten, die für Basic wahrlich traumhaft sind. Der Speicherbedarf ist dabei minimal, er liegt nur noch unwesentlich über der Länge vergleichbarer Maschinenprogramme.
Dafür muß man allerdings folgende Einschränkungen in Kauf nehmen, über welche jedoch die sensationellen Ergebnisse hinwegtrösten (der Zweck heiligt manchmal die Mittel!):

- Pro Zeile darf nur ein einziger Befehl benutzt werden.
- Zulässige Befehle sind LET, REM, CLR, END, GOTO, GOSUB, RETURN, SYS, PRINT, POKE, IF...THEN, STOP.
- LET ist keine optionale Anweisung, sondern vor jeder Variablenzuweisung verpflichtend.
- Pro LET-Befehl ist nur eine Rechenoperation zulässig; Kettenrechnungen oder Klammern sind nicht möglich. Beispielsweise muß die Zeile »10 LET A=4*X+1« ersetzt werden durch »10 LET A=4*X« und »11 LET A=A+1«.
- In allen anderen Befehlen dürfen keine Rechenoperationen vorkommen; nur Variable oder positive Zahlen sind erlaubt.
- Erlaubte Rechenoperationen sind nur »+«, »-«, »*«, »/«, »AND« und »OR«.
- Zulässige Variablennamen sind nur die Buchstaben »A« bis »Z«; alle Variablen sind ganzzahlig im Bereich 0–65535.
- Als Funktionen sind nur PEEK und RND vorhanden. Bei RND() muß immer 255 als Argument angegeben werden, woraufhin Zufallszahlen zwischen 0 und 255 geliefert werden.
- Hinter PRINT darf nur ein Variablenname oder ein Text in Anführungszeichen folgen. Rechnungen, Funktionen oder numerische Konstanten sind nicht zulässig.
- Bei IF sind nur Vergleiche wie »=«, »<« oder »>« erlaubt. Der erste Vergleichsoperand muß eine Variable sein, der zweite entweder eine Variable oder ein feststehender Integerwert.
- Hinter THEN darf nur eine Zeilennummer folgen, weitere Befehle sind unzulässig.

Zugegeben, die Programmierung für den Ascompiler 64 ist extrem gewöhnungsbedürftig und im Vergleich zum normalen Basic umständlich und wenig komfortabel. Im Vergleich zur Programmierung in Assembler, die von der erreichten Geschwindigkeit her die einzige Alternative wäre, ist der Ascompiler 64 jedoch der reinste Luxus. Es liegt nur noch an Ihnen, ob Sie die dadurch gegebenen Möglichkeiten nutzen wollen. Sie haben mit diesem 64'er-Listing, welches auch auf einer Programmdiskette zum Heft 1/86 erhältlich ist, auf jeden Fall ein preiswertes System an der Hand, mit dem schnelle Action-Spiele und andere ehedem typische Assembler-Anwendungen in Basic zu realisieren sind.

Da der Ascompiler 64 ein sehr unkonventioneller Compiler ist, wollte ich ihn im Rahmen dieses Buches vorstellen; vielleicht interessiert sich ja der eine oder andere Leser für dieses revolutionäre System.

7 Menüprogrammierung in Basic oder: Das Auge ißt (tippt) mit

Viele Programme zeichnen sich dadurch aus, daß sie eine sehr anwenderfreundliche Menüsteuerung besitzen. Nach zwei Hauptkriterien werden die Menüs, welche ja den Großteil des optischen Erscheinungsbildes eines Programms ausmachen, bewertet:

- Wie bequem sind sie zu bedienen?
- Wie gering ist die Wahrscheinlichkeit eines Eingabefehlers?

7.1 Menüs über Cursortasten

Sieht man sich einige Menüs von Programmen kommerzieller Software-Hersteller an, wie zum Beispiel Textverarbeitungs- oder Dateiverwaltungsprogramme, so kann man hier mit den Cursortasten, welche dann ein inverses Feld bewegen, die einzelnen Programmfunktionen auswählen; durch <RETURN> wird der jeweils hervorgehobene Menüpunkt aktiviert. Dies garantiert nicht nur hohen Bedienungskomfort, sondern schließt auch in weitem Maße Fehlbedienungen aus: Das Inversfeld zeigt den aktuellen Menüpunkt deutlich an.

7.1.1 Einfache Menüs über Auswahltasten

Die einfachste Form der Menüsteuerung, die vor allem von Anfängern programmiert wird, ist die Durchnumerierung der einzelnen Menüpunkte. Mit der Basic-Anweisung ON...GOTO lassen sich diese Unterprogramme schnell aufrufen. Beispiel:

```
100 INPUT "MENUEPUNKT";A
110 ON A GOTO 1000,2000,3000,...,n
```

Etwas mehr Komfort bringt die Verwendung von GET, die dem Anwender das Drücken von <RETURN> erspart:

```
100 PRINT "MENUEPUNKT?"
105 POKE 198,0:WAIT 198,1:GET A$:REM auf Taste warten, dann A$ einlesen
110 ON VAL(A$) GOTO 1000,2000,3000,...,n
```

Eine solche Menüsteuerung wird in Listing 7.1 angewandt.

Listing 7.1: MENUE-BSP. 1

```
100 if peek(769)<>243 then print"erst paradoxon basic laden":stop
110 :
120 :
130 rem ******************************
140 rem *                            *
150 rem *  menue-beispielprogramm #1 *
160 rem *                            *
170 rem *    anwendungsbeispiel zur  *
180 rem *  einfachsten menuesteuerung *
190 rem *                            *
200 rem ******************************
210 rem *                            *
220 rem * written 12.10.1987 by      *
230 rem *                            *
240 rem *      florian mueller       *
250 rem *                            *
260 rem ******************************
270 :
280 :
290 :
300 print"(clr)"
310 print( 0,6)"diskettenverwaltungsprogramm"
320 print( 1,6)"============================"
330 print( 5,5)"(rvs) 1 (off) directory"
340 print( 7,5)"(rvs) 2 (off) disk-befehl senden"
350 print( 9,5)"(rvs) 3 (off) disk-status abfragen"
360 print(11,5)"(rvs) 4 (off) programm beenden"
370 :
400 get a$:if a$<"1" or a$>"4" then 400
410 on val(a$) goto 1000,1500,2000,2500
420 :
430 :
1000 rem *** menuepunkt 1 ***
1010 rem *** (directory)  ***
1020 :
1030 print"(clr)(down)"
1040 @$
1050 print
1060 print(24,14)"(rvs)taste druecken";
1070 get a$:if a$="" then 1070
1080 goto 300
1090 :
1100 :
1500 rem *** menuepunkt 2 ***
1510 rem *** (diskbefehl) ***
1520 :
1530 print"(clr)"
1540 print"disk-befehl: ";
1550 poke 198,1:poke 631,34:rem anfuehrungszeichen als erstes eingabezeichen
1560 rem                        ermoeglicht eingabe der trennzeichen ("," etc.)
1570 open 1,0:input#1,db$:close 1:print
1580 open 1,8,15,db$:close 1:print
1590 goto 300
1600 :
1610 :
2000 rem *** menuepunkt 3 ***
2010 rem *** (diskstatus) ***
2020 :
2030 print"(clr)"
2040 print"(clr)(down)(down)disk-status: ";:@
2050 goto 1050
2060 :
2500 rem *** menuepunkt 4 ***
2510 rem ***   (beenden)   ***
```

```
2520 :
2530 print"(clr)auf wiedersehen!"
2540 end
```

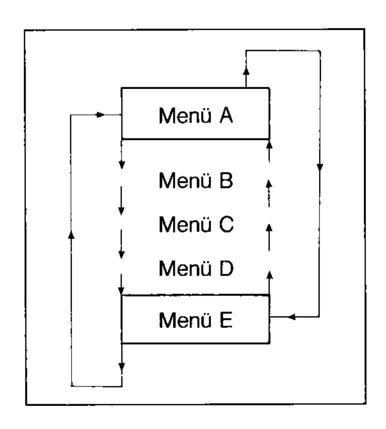
Dieses rudimentäre Diskettenhilfsprogramm ermöglicht die Anzeige des Disketteninhalts (Directory), das Senden eines Diskettenkommandos, die Abfrage des Diskettenstatus und das Verlassen des Programms. Über GET (Zeile 400) wird die Nummer des gewünschten Menüpunktes eingegeben, auf Richtigkeit der Eingabe mit IF...THEN geprüft (es sind nur <1>, <2>, <3> und <4> als Eingaben zugelassen) und in Zeile 410 unmittelbar in den entsprechenden Programmteil gesprungen; die Zahlen hinter ON...GOTO zeigen jeweils auf die erste Zeilennummer eines Programmteils. Die Routinen zu den einzelnen Menüpunkten belegen die Zeilen 1000–2540, also den überwiegenden Teil des Programms.
Diese Form des Menüs ist allerdings noch recht unkomfortabel. Vor allem stört, daß man zu einer gewünschten Funktion zuerst die richtige Nummer eingeben muß. Und mehr als 9 oder (wenn ein Menüpunkt die Nummer 0 bekommt) maximal 10 Menüpunkte lassen sich nicht mehr mit Ziffern zuordnen, es müßte also auf Buchstaben oder den INPUT-Befehl zur Eingabe zweistelliger Menüpunkt-Nummern ausgewichen werden.

7.1.2 Beispiel-Menü über Cursortasten

Damit Sie gleich in den Genuß eines guten Menüs kommen, laden Sie das Programm »MENUE-BSP. 2«, mit dem wir uns noch eine Weile beschäftigen werden, und starten Sie es. Zuvor ist jedoch, wie bei allen noch folgenden Beispielen in diesem Kapitel, das Programm »Paradoxon Basic« (Kapitel 5) zu laden.
An der Bildschirmanzeige erkennen Sie bestimmt, daß die Menüpunkte intern mit den Zahlen 0–4 durchnumeriert werden, weshalb diese Zahlen jeweils in Klammern hinter dem betreffenden Menüpunkt stehen. Mit folgenden Tasten bedienen Sie das Menü:

<CRSR DOWN>	Inversfeld nach unten
<CRSR UP>	Inversfeld nach oben
<HOME>	Inversfeld in oberste Position
<RETURN>	Punkt an Inversfeld-Position auswählen
<CTRL C>	Programm abbrechen
(andere Tasten sind wirkungslos)	

Wenn dabei das Inversfeld, das sozusagen der »Menü-Cursor« ist, mit <CRSR UP> über den obersten Punkt (Menüpunkt A) hinausbewegt wird, so kommt man zum untersten Menüpunkt, also Punkt E; ähnlich ist es, wenn mit <CRSR DOWN> das Inversfeld über den Menüpunkt E bewegt wird, was auf den obersten Punkt (Punkt A) führt. Dies zeigt auch Bild 7.1.



Probieren Sie ohne weiteres alle Funktionen aus. Sollte Ihnen diese Art der Menüsteuerung gefallen, dann erfahren Sie im folgenden mehr über deren Aufbau.

Bild 7.1: Schemazeichnung (die Pfeile zeigen die Bewegungsrichtung des Inversfeldes an)

7.1.3 Funktionsweise des komfortablen Beispielmenüs

Listing 7.2: MENUE-BSP. 2

```
100 if peek(769)<>243 then print"erst paradoxon basic laden":stop
110 :
120 :
130 rem ********************************
140 rem *                              *
150 rem *  menue-beispielprogramm #2   *
160 rem *                              *
170 rem *  erklaerungshilfe und demo   *
180 rem * zur besseren menuesteuerung  *
190 rem *                              *
200 rem ********************************
210 rem *                              *
220 rem * written 12.10.1987 by        *
230 rem *                              *
240 rem *      florian mueller         *
250 rem *                              *
260 rem ********************************
270 :
280 :
290 :
300 rem >>> initialisierung <<<
310 :
320 clr:rem variablen sicherheitshalber loeschen, um fehlerfreies funktionieren
330 rem     von zeile 350 zu gewahrleisten (>>redim'd array error<< vermeiden)
340 a = 5:rem fuenf menuepunkte
350 dim rt (a-1):rem revers-tabelle dimensionieren
360 p = 0:rem menuepunkt 0 (vor-)einstellen
370 rt(p) = 1:rem aktueller menuepunkt (0) revers
380 print"(clr)":poke 53280,0:poke 53281,0
390 :
400 rem >>> ausgabe der menuepunkte <<<
410 :
420 print(0,3)"(cyn)menuesteuerung - beispielprogramm"
430 print(1,3)"=================================="
440 :
450 poke 199,rt(0):print( 6,9)"(yel)menuepunkt a(off) (0)"
460 poke 199,rt(1):print( 8,9)"(grn)menuepunkt b(off) (1)"
470 poke 199,rt(2):print(10,9)"(pur)menuepunkt c(off) (2)"
480 poke 199,rt(3):print(12,9)"(lred)menuepunkt d(off) (3)"
```

```
490 poke 199.rt(4):print(14,9)"(gry3)menuepunkt e(off) (4)"
500 print"(down)(down)(down)(wht)aktuell angewaehlter menuepunkt: ";chr$(65+p);"
 /";p
510 :
520 rem >>> tastaturabfrage <<<
530 :
540 get t$:if t$="" then 540
550 if t$=chr$(3) then print"(down)(down)programmabbruch.":end:rem <control c>
560 if t$=chr$(13) then on p+1 goto 1000,2000,3000,4000,5000
570 rem
580 rem    nun werden alle diejenigen tasten behandelt, die das inversfeld
590 rem    verschieben
600 rem
610 rt(p) = 0:rem aktuelle position des reversfeldes aufheben
620 if t$=chr$(17) then p = p+1 + a*(p=(a-1)):rem <cursor down>
630 if t$=chr$(145)then p = p-1 - a*(p=0):rem <cursor up>
640 if t$=chr$(19) then p = 0:rem <home>
650 rt(p) = 1:rem neue position des inversfeldes steht in p, also wird hier das
660 rem            feld rt() entsprechend p aktualisiert (siehe auch zeile 610)
670 goto 450
680 :
690 :
700 rem >>> es folgen die routinen <<<
710 rem >>>  zu den menuepunkten.  <<<
720 :
1000 print"(clr)sie haben den obersten menuepunkt ange-"
1010 print:print"waehlt. interne nummer:"p
1020 print(20,10)"(rvs)bitte eine taste druecken"
1030 get t$:if t$="" then 1030
1040 goto 380
2000 print"(clr)sie haben den zweitobersten menuepunkt"
2010 print:print"angewaehlt. interne nummer:"p
2020 goto 1020
3000 print"(clr)sie haben den mittleren menuepunkt ange-"
3010 print:print"waehlt. interne nummer:"p
3020 goto 1020
4000 print"(clr)sie haben den zweituntersten menuepunkt"
4010 print:print"angewaehlt. interne nummer:"p
4020 goto 1020
5000 print"(clr)sie haben den untersten menuepunkt an-"
5010 print:print"gewaehlt. interne nummer:"p
5020 goto 1020
```

Die Variablen

A — enthält die Anzahl der Menüpunkte (5). A ist eine Konstante, da sich der Wert im gesamten Programmverlauf nicht ändert.

RT() — ist ein Array, das für jeden Menüpunkt die Information enthält, ob er hervorgehoben werden soll. Ist RT(n)=1, so wird der Menüpunkt »n« invertiert, d.h. auf ihm liegt das Inversfeld; andernfalls ist RT(n)=0.

P — beinhaltet die Nummer des derzeit angewählten Menüpunktes, d.h. des Menüpunktes, auf welchem das Inversfeld liegen soll. P wird durch die Cursortasten geändert und schließlich zur Auswahl der Unterprogramme herangezogen.

T$ — enthält jeweils den letzten eingelesenen Tastendruck und ist somit eine Hilfsvariable für Vergleichsoperationen.

Um die Bedeutung der Variablen besser zu verstehen, sei nur aufs Gröbste skizziert, wie das Programm mit den Variablen umgeht: Bei jeder Inversfeld-Bewegung wird das Array RT() gemäß dem neu einzustellenden Wert von P berechnet, so daß bei der sofort anschließenden Bildschirmausgabe das Menü in gewünschter Form erscheint. Bei Auslösen des Menüpunktes enthält P den entsprechenden Index für den Sprung in die einzelnen Menüroutinen.

Das Programm

Nun zur zeilenorientierten Betrachtung.

Zeile 100: Test auf Paradoxon Basic

Hier vollzieht sich die Prüfung, ob Paradoxon Basic bereits aktiviert wurde. Dazu wird das High-Byte des Warmstart-Vektors mit dem unter Paradoxon Basic bestehenden Wert verglichen.

Zeilen 300–390: Initialisierung

Zunächst werden die Variablen vordefiniert (320–370) und dann die Bildschirmfarben gesetzt (380). Bei der Variablendefinition bekommt P (Programmpunkt) den Wert Null zugewiesen, womit der oberste Menüpunkt (Punkt A) als Ausgangsstellung gilt. Dieser wird auch wegen des RT(0)-Inhalts in Zeile 450 hervorgehoben. Die anderen Elemente von RT() sind nach Zeile 350 mit 0 belegt (0 bedeutet dabei »nicht revers«). In Zeile 380 nimmt der Bildschirm schwarze Farbe an.

Zeilen 400–510: Ausgabe der Menüpunkte

Bei Ausgabe eines Menüpunktes wird jeweils eine individuelle Schriftfarbe gesetzt, da ein mehrfarbiges Menü übersichtlicher ist als ein einfarbiges; auf Schwarzweiß-Monitoren erscheint es in unterschiedlicher Helligkeit, was ebenfalls zur Übersichtlichkeit beiträgt.
Die Menüpunkt-Texte werden über PRINT ausgegeben, wobei hier die Positionierung über x, y erfolgt, die nur unter Paradoxon Basic möglich ist. Vor der PRINT-Ausgabe entscheidet ein Befehl wie »POKE 199,RT(n)« über die eventuelle Invertierung der Bildschirmausgabe (siehe 2.5): Steht in Adresse 199 der Wert 0, so wird die nächste PRINT-Anweisung normal ausgeführt, bei Werten ab 1 automatisch invertiert. Dies hält nur so lange an wie das Steuerzeichen CHR$(18) (= RVS ON), welches durch den POKE-Befehl nur simuliert wird.
Die PRINT-Befehle stehen – soweit möglich – untereinander; am Bildschirm erscheinen die Farbsteuerzeichen ebenfalls in gleicher Länge, wodurch das äußere Erscheinungsbild noch mehr gewinnt. Dies wird dadurch erzielt, daß einstelligen Ziffern in der Koordinatenangabe ein Leerzeichen vorangestellt wird; das ist optisch nicht nur schöner, sondern erleichtert vor allem das Editieren dieser Zeilen, weil man sich an den unmittelbar darüber- und/oder darunterstehenden Zeilen orientieren kann.

In Zeile 500 wird informationshalber der derzeit angewählte Menüpunkt angezeigt, was natürlich für den reinen Anwender als Information über die programmtechnische Funktionsweise belanglos ist. In der Endform des Menüs entfällt Zeile 500 also.

Zeilen 520–670: Tastaturbehandlung

Dieser Programmteil fragt die Tastatur ab und bewegt das Inversfeld entsprechend; bei <RETURN> wird ein Menüpunkt ausgeführt (Zeile 560), bei <CTRL C> abgebrochen (Zeile 550). Die Inversfeld-Bewegung geschieht, indem zunächst in Zeile 610 das einzige Element mit dem Wert 1 ebenfalls gelöscht wird; jetzt befindet sich das Feld RT() komplett im Ausgangszustand. Die Zeilen 620–640 berechnen nun eine neue Inversfeld-Position P, gemäß deren Wert in Zeile 650 das entsprechende Element von RT() den Wert 1 erhält. Gewissermaßen »wandert« also die 1-Belegung von der alten zur neuen Position; Näheres dazu später. In Zeile 670 erfolgt ein Rücksprung zur Ausgabe des Menüs, wobei die Menüpunkte neu gedruckt werden und wieder die Tastaturabfrage eintritt.

Zeilen 700–Ende: Routinen zu den Menüpunken

Zu jedem Menüpunkt ist eine eigene Routine vorhanden, welche letztlich wieder zu Zeile 1020 verzweigt.

Bewegung des Inversfeldes – ein simpler Trick

Um die Bewegung des Inversfeldes zu simulieren, muß das alte Bild bei jedem Tastendruck, der die Inversfeld-Position beeinflußt, mit dem neuen Bildschirmaufbau überschrieben werden. Die Frage stellt sich von selbst: Wie erreicht man es, daß sich die Bilder genau decken? Die Antwort ist denkbar einfach: Da wir den PRINT-Befehl mit dem Paradoxon-Basic-Zusatz der Koordinatenangabe versehen, werden die Texte, welche sich ja im eigentlichen Sinn nicht ändern (nur die Reversdarstellung wechselt), immer an denselben Bildschirmpositionen ausgegeben. Da nun das Inversfeld immer nur auf einem einzigen Menüpunkt liegt, werden alle Punkte bis auf einen in Normaldarstellung ausgegeben. Nur an zwei Stellen ergeben sich Veränderungen, die als »Bewegung« erscheinen:

1. Der Punkt, auf dem das Inversfeld vorher lag, wird nicht mehr revers abgedruckt. Dadurch entsteht der Eindruck, das Inversfeld hätte sich von dort wegbewegt.
2. Ähnlich verhält es sich mit dem Punkt, auf welchen das Inversfeld gesteuert wurde: Dieser wird jetzt revers gedruckt und man meint, das Inversfeld hätte sich dorthin bewegt.

Dieser Vorgang wird Ihnen sicher deutlicher, wenn Sie testhalber Zeile 670 modifizieren:

```
670 PRINT CHR$(147):GOTO 450
```

Dann wird der Neuaufbau des Bildes sichtbar, weil der Bildschirm jedesmal komplett gelöscht wird, bevor die Menüpunkte erneut erscheinen. Dies macht sich dann durch ein Flackern des

Bildes bemerkbar, welches zur Demonstration recht auffällig ist. Ein weiterer Effekt: Die Kopfzeilen erscheinen nur einmal, nach der nächstbesten Inversfeld-Bewegung jedoch nicht mehr, da Zeile 670 den Bildschirm löscht und dann in Zeile 450 verzweigt.

Die Steuerung des Inversfeldes

Wie wir wissen, hängt die Position des Inversfeldes davon ab, welche Werte im Array RT() und in der Variablen P stehen. Ebenfalls ist bekannt, daß vor den IF-Abfragen das Inversfeld entfernt wird (Zeile 610) und nach Neupositionierung in Zeile 650 eine Neubelegung erfolgt. Uns interessiert nun, wie ein neuer Wert von P zwischen den Zeilen 610 und 650 errechnet wird, der sich aus der gewünschten Bewegung <CRSR UP>, <CRSR DOWN> oder <HOME> ergibt.

So funktioniert <HOME>

Am einfachsten ist dies bei <HOME>, welches den ASCII-Code CHR$(19) hat:

```
640 IF T$=CHR$(19) THEN P=0
```

Wenn <HOME> gedrückt wurde, so soll also der oberste Menüpunkt angesprungen werden, dessen Nummer 0 hier an P zugewiesen wird. An diesem Beispiel kann man den Ablauf gut verfolgen, hier sei er noch einmal zusammengefaßt:

```
610 RT(P) = 0:REM Inversfeld zunächst löschen
620 IF T$=CHR$(17) ...trifft bei <HOME> nicht zu
630 IF T$=CHR$(145)...trifft bei <HOME> nicht zu
640 IF T$=CHR$(19) THEN P=0:REM Menüpunkt 0 setzen
650 RT(P) = 1:REM Inversfeld auf Zielpunkt P (im Beispiel 0) setzen
660 REM
670 GOTO 450:REM neuer Bildschirmaufbau
```

Bei den Tasten <CRSR DOWN> und <CRSR UP> verläuft dies auf gleiche Weise, allerdings wird P auf andere Weise neu berechnet, da es sich schließlich auch um andere Funktionen handelt.

So funktioniert <CRSR DOWN>

Bei <CRSR DOWN> wird Zeile 620 aktiv:

```
620 IF T$=CHR$(17) THEN P = P+1 + A*(P=(A-1))
```

Da die IF-Bedingung bei Drücken von <CRSR DOWN> erfüllt ist, beschränken wir uns fortan auf die Besprechung des THEN-Teils. Dazu müssen Sie jedoch Kenntnis von 2.6 haben, wo die numerische Auswertung logischer Ausdrücke beschrieben steht.

```
... p = p+1 + a*(p=(a-1))
```

Zunächst wird also P um 1 erhöht, da auch die weiter unten stehenden Menüpunkte höhere Kennziffern haben. Dafür ist der Anfangsausdruck »P = P+1« zuständig. Diesem folgt ein recht komplizierter Ausdruck »+ A*(P=(A–1))«, der bei mathematischer Auflösung jedoch an Übersichtlichkeit zunimmt. Setzen wir dazu den Wert 5 für die Konstante A ein:

```
... p = p+1 + 5*(p=(5—1))
```

Etwas weitergerechnet, ergibt sich:

```
... p = p+1 + 5*(p=4)
```

Jetzt stört uns nur noch ein Klammerausdruck, der aus einer logischen Abfrage besteht. Wenn P=4 unwahr ist, also P nicht auf dem untersten Menüpunkt liegt, bekommt die Klammer den Wert 0:

bei P<>4: ... P = P+1 + 5*0
bei P<>4: ... P = P+1 + 0
bei P<>4: ... P = P+1

Dies läßt sich einfach erklären: Das Inversfeld wird im Normalfall um 1 nach unten bewegt (also P um 1 erhöht), wenn <CRSR DOWN> betätigt wird. Der Sonderfall P=4 (Inversfeld auf unterstem Menüpunkt) löst bekanntlich den Sprung zum obersten Menüpunkt (Punkt 0) aus; auch dies wird von unserer Formel korrekt ausgeführt:

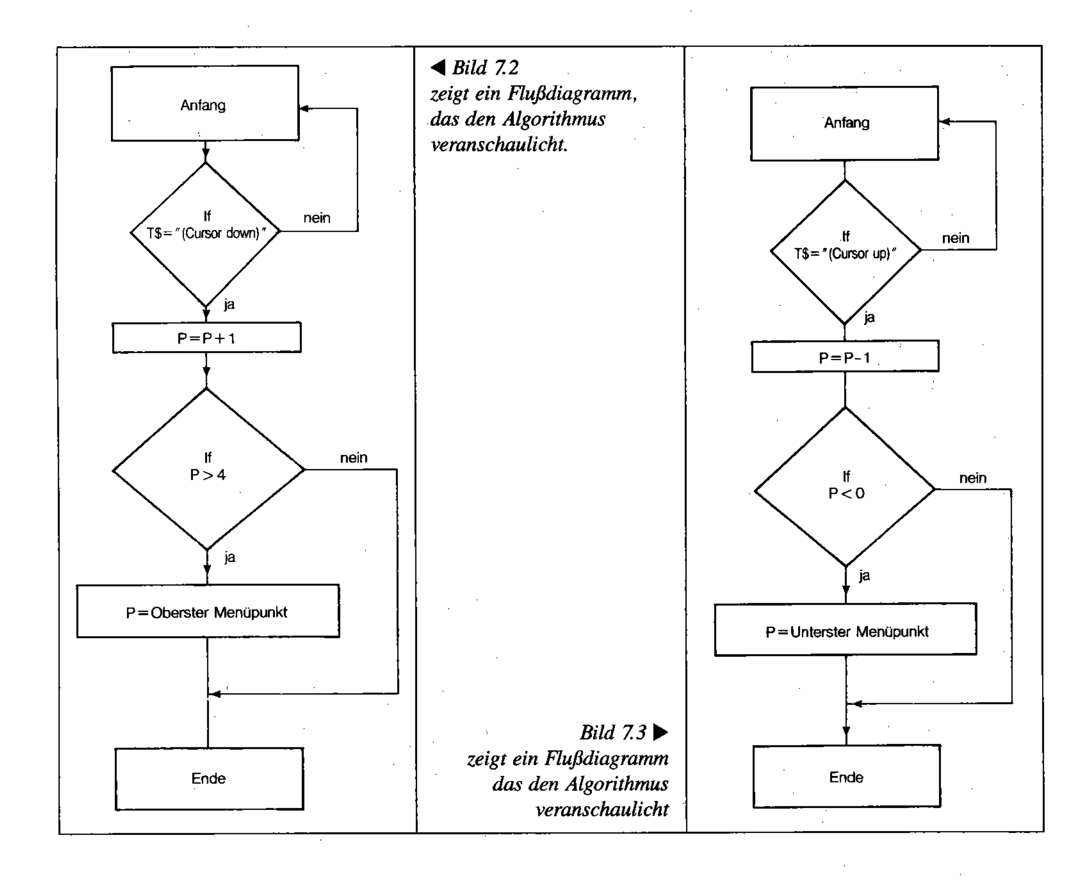
bei P=4: ... P = P+1 + 5*(–1)
bei P=4: ... P = P+1 + (–5)
bei P=4: ... P = P+1 –5
bei P=4: ... P = P–4

Da P=4 vorausgesetzt ist, können wir nun auch auf der rechten Seite der Zuweisung für P den Wert 4 einsetzen (den Fall, daß P <> 4 ist, haben wir bereits behandelt):

bei P=4: ... P = 4–4
bei P=4: ... P = 0

Fassen wir noch einmal zusammen:

Formel in Zeile 620:		P = P+1 + A*(P=(A–1))
nach Einsetzen von A:		P = P+1 + 5*(P=4)
wenn P=4 falsch ist:		P = P+1 + 5*0
daher	=>	P = P+1
wenn P=4 wahr ist:		P = P+1 + 5*(–1)
daher bei Einsetzen von P:		P = 4–4
	=>	P = 0



◀ *Bild 7.2 zeigt ein Flußdiagramm, das den Algorithmus veranschaulicht.*

Bild 7.3 ▶ *zeigt ein Flußdiagramm das den Algorithmus veranschaulicht*

Eine entsprechende Konstruktion mit IF wäre folgendermaßen zu formulieren:

```
620 IF T$<>CHR$(17) THEN 630
621 :IF P<>4 THEN P=P+1:GOTO 630
622 :P=0
630 ...
```

Dies ist zwar einsichtiger als unsere 1-Zeilen-Lösung, dafür aber auch etwas länger und langsamer. Ansonsten spricht aber nichts dagegen, daß Sie in Ihren Programmen mit einer solchen IF-THEN-Konstruktion arbeiten, wenn Ihnen die lange Formel zu kompliziert erscheint. Dann wäre allerdings auch die IF-THEN-Formulierung noch zu verbessern:

```
620 IF T$=CHR$(17) THEN P=P+1:IF P>4 THEN P=0
```

Hier wird zunächst addiert und dann erst festgestellt, ob durch die Addition ein unerlaubter Wert entstanden ist.

So funktioniert <CRSR UP>

Betrachtet man Zeile 630, so sieht man starke Ähnlichkeiten zu Zeile 620. Die Situation von <CRSR UP> ist die, daß das Inversfeld um eine Position nach oben bewegt werden muß; dazu ist nur P um 1 herunterzuzählen (P=P–1), weil weiter oben stehende Menüpunkte niedrigere Kennziffern haben. Wird aber vom obersten Menüpunkt (Kennziffer 0) aus eine CRSR-UP-Bewegung durchgeführt, soll der unterste Menüpunkt angesprungen werden. Die Berechnung steht in besagter Zeile 630, wobei uns wieder nur der THEN-Teil interessiert:

```
... p = p-1 - a*(p=0)
... p = p-1 - 5*(p=0)
```

Jetzt sind wieder zwei Fälle zu betrachten:

```
P=0 ist wahr:        ... P = P-1 - 5*(-1)
                     ... P = P-1 - (-5)
                     ... P = P-1 + 5
                     ... P = P + 4
P eingesetzt:        ... P = 0+4
                     ... P = 4
```

Die Nummer des untersten Menüpunktes ist genau 4.

```
P=0 ist falsch:      ... P = P-1 - 5*0
                     ... P = P-1 - 0
                     ... P = P-1
```

Solange P=0 falsch ist und sich somit das Inversfeld nicht auf dem obersten Menüpunkt befindet, muß nur 1 subtrahiert werden. Bild 7.3 zeigt noch einmal die Vorgehensweise des Programms.

Auch hier wollen wir nun die umständlich erscheinende Schreibweise durch eine IF-THEN-Konstruktion ersetzen:

```
630 IF T$< >CHR$(145) THEN 640
631 :IF P< >0 THEN P=P-1:GOTO 640
632 :P=4
640 ...
```

Oder einfacher:

```
630 IF T$< >CHR$(145) THEN P=P-1:IF P<0 THEN P=4
```

Ich nehme an, daß Ihnen das Prinzip jetzt klar ist. Sollten Sie dennoch Schwierigkeiten mit den raffinierten Formeln haben, können Sie in Ihren Programmen auch die vorgestellten Lösungen mit IF...THEN verwenden, die ja schon mit Grundkenntnissen in Basic zu realisieren sind.

7.1.4 Optimierte Menüsteuerung

Wir werden uns jetzt mit der Anwendung dieser Menüsteuerung in der Praxis befassen und einige Änderungen sowie Ergänzungen vornehmen. Schließlich wird noch eine universelle Routine vorgestellt, zu deren Gebrauch überhaupt kein Verständnis der Funktionsweise erforderlich ist. Als erstes wollen wir das Menüprogramm optimieren, indem

- REM-Kommentare entfernt werden
- Zeilen komprimiert (zusammengefaßt) werden
- die Konstante A durch den Zahlenwert 5 ersetzt wird

Dadurch wird das Programm sowohl kürzer als auch schneller.

Listing 7.3: MENUE-BSP. 3

```
100 if peek(769)<>243 then print"erst paradoxon basic laden":stop
110 :
120 :
130 rem ********************************
140 rem *                              *
150 rem *  menue-beispielprogramm #3   *
160 rem *                              *
170 rem *     optimiertes beispiel     *
180 rem * zur besseren menuesteuerung  *
190 rem *                              *
200 rem ********************************
210 rem *                              *
220 rem * written 12.10.1987 by        *
230 rem *                              *
240 rem *       florian mueller        *
250 rem *                              *
260 rem ********************************
270 :
280 :
290 :
300 clr:dim rt(4):p=0:rt(p)=1
310 poke53280,0:poke53281,0:print"(clr)"
320 print(0,3)"(cyn)menuesteuerung - beispielprogramm"
330 print(1,3)"================================="
340 poke199,rt(0):print( 6,9)"(red)menuepunkt a(off)  (0)"
350 poke199,rt(1):print( 8,9)"(grn)menuepunkt b(off)  (1)"
360 poke199,rt(2):print(10,9)"(lblu)menuepunkt c(off)  (2)"
370 poke199,rt(3):print(12,9)"(yel)menuepunkt d(off)  (3)"
380 poke199,rt(4):print(14,9)"(gry3)menuepunkt e(off)  (4)(wht)"
390 get t$:if t$="" then 390
400 if t$=chr$(13) then on p+1 goto 480,530,560,590,620
410 if t$=chr$(3) then print"(down)(down)programmabbruch.":end
420 rt(p)=0
430 if t$=chr$(17) then p=p+1+5*(p=4)
440 if t$=chr$(145)then p=p-1-5*(p=0)
450 if t$=chr$(19) then p=0
460 rt(p)=1
470 goto 320
480 print"(clr)sie haben den obersten menuepunkt ange-"
490 print"(down)waehlt. interne nummer:"p
500 print(24,15)"(rvs)bitte eine taste";
510 get t$:if t$="" then 510
520 goto 300
530 print"(clr)sie haben den zweitobersten menuepunkt"
```

```
540 print"(down)angewaehlt. interne nummer:"p
550 goto500
560 print"(clr)sie haben den mittleren menuepunkt an-"
570 print"(down)gewaehlt. interne nummer:"p
580 goto500
590 print"(clr)sie haben den zweituntersten menuepunkt"
600 print"(down)angewaehlt. interne nummer:"p
610 goto500
620 print"(clr)sie haben den untersten menuepunkt ange-"
630 print"(down)waehlt. interne nummer:"p
640 goto500
```

Jetzt belegt das eigentliche Programm noch nicht einmal die Hälfte des vorher erforderlichen Speicherplatzes, wenn man auch noch den REM-Vorspann (Zeilen 130–260) entfernt. Ich habe mich aber für diesen Grad der Optimierung entschieden, damit noch eine möglichst große Ähnlichkeit zur ursprünglichen Version (Listing 7.2) besteht und Sie leichter vergleichen können. Das Ersetzen der Variablen A durch den jeweiligen Zahlenwert erschwert zwar ein späteres Erweitern um weitere Menüpunkte oder das Streichen einer Option, aber als Beispiel sei ein Menü mit 4 (statt 5) Menüpunkten vorgestellt:

Listing 7.4: MENUE-BSP. 4

```
100 if peek(769)<>243 then print"erst paradoxon basic laden":stop
110 :
120 :
130 rem ********************************
140 rem *                              *
150 rem *  menue-beispielprogramm #4   *
160 rem *                              *
170 rem *     optimiertes beispiel     *
180 rem *   fuer diskettenverwaltung   *
190 rem *                              *
200 rem ********************************
210 rem *                              *
220 rem * written 12.10.1987 by        *
230 rem *                              *
240 rem *       florian mueller        *
250 rem *                              *
260 rem ********************************
270 :
280 :
290 :
300 clr:dim rt(3):p=0:rt(p)=1
310 poke53280,0:poke53281,0:print"(clr)"
320 print(0,10)"(cyn)diskettenverwaltung"
330 print(1,10)"=================="
340 poke199,rt(0):print( 6,14)"(red)directory"
350 poke199,rt(1):print( 8,13)"(grn)disk-befehl"
360 poke199,rt(2):print(10,13)"(lblu)disk-status"
370 poke199,rt(3):print(12,11)"(yel)programm beenden"
380 get t$:if t$="" then 380
390 if t$=chr$(13) then on p+1 goto 470,580,700,770
400 if t$=chr$(3) then on p+1 goto 820,900,1010,1100
410 rt(p)=0
420 if t$=chr$(17) then p=p+1+4*(p=3)
430 if t$=chr$(145)then p=p-1-4*(p=0)
440 if t$=chr$(19) then p=0
```

```
450 rt(p)=1
460 goto 320
470 rem *** menuepunkt 1 ***
480 rem *** (directory)  ***
490 :
500 print"(clr)(down)"
510 @$
520 print
530 print(24,14)"(rvs)taste druecken";
540 get a$:if a$="" then 540
550 print"(clr)":goto 320
560 :
570 :
580 rem *** menuepunkt 2 ***
590 rem *** (diskbefehl) ***
600 :
610 print"(clr)"
620 print"disk-befehl: ";
630 poke 198,1:poke 631,34:rem anfuehrungszeichen als erstes eingabezeichen
640 rem                        ermoeglicht eingabe der trennzeichen ("," etc.)
650 open 1,0:input#1,db$:close 1:print
660 open 1,8,15,db$:close 1:print
670 goto 310
680 :
690 :
700 rem *** menuepunkt 3 ***
710 rem *** (diskstatus) ***
720 :
730 print"(clr)"
740 print"(clr)(down)(down)disk-status: ";:@
750 goto 520
760 :
770 rem *** menuepunkt 4 ***
780 rem ***   (beenden)   ***
790 :
800 print"(clr)auf wiedersehen!"
810 end
820 print"(clr)(down)(rvs)informationstext zu >>directory<<"
830 print"(down)(down)dieser menuepunkt gibt das inhalts-"
840 print"(down)verzeichnis der aktuell eingelegten"
850 print"(down)diskette mit atemberaubendem tempo aus."
860 print"(down)anschliessend muessen sie eine taste"
870 print"(down)druecken, woraufhin sie sich wieder"
880 print"(down)im hauptmenue befinden."
890 goto 530
900 print"(clr)(down)(rvs)informationstext zu >>disk-befehl<<"
910 print"(down)(down)damit ist es ihnen moeglich, befehle"
920 print"(down)an das diskettenlaufwerk zu senden."
930 print"(down)vor ihrer eingabe erscheint ein anfueh-"
940 print"(down)rungszeichen, von dem sie sich nicht"
950 print"(down)stoeren lassen sollten; es ist lediglich"
960 print"ein trick, die eingabe von (rvs)allen(off) zeichen"
970 print"zuzulassen (auch >>,<<, >>:<<, >>;<<)."
980 print"(down)die syntax der floppy-befehle entnehmen"
990 print"(down)sie bitte dem handbuch zur 1541/70/71."
1000 goto530
1010 print"(clr)(down)(rvs)informationstext zu >>disk-status<<"
1020 print"(down)(down)wenn die rote leuchtdiode an ihrem lauf-"
1030 print"werk nervoes blinkt, so liegt irgendein"
1040 print"(down)grosser oder kleiner fehler vor."
1050 print"(down)genauere auskunft erteilt dieser menue-"
1060 print"(down)punkt, der eine fehlernummer und eine"
1070 print"(down)dazugehoerige meldung sowie die angabe"
1080 print"(down)von spur und sektor darstellt."
1090 goto530
1100 print"(clr)(down)(rvs)informationstext zu >>programm beenden<<"
1110 print"(down)sollten sie tatsaechlich einmal dieses"
```

```
1120 print"(down)wundervollen programms ueberdruessig"
1130 print"(down)sein oder sich selbst ans werk machen"
1140 print"(down)wollen, verlassen sie mit dem letzten"
1150 print"(down)menuepunkt kurz und schmerzlos dieses"
1160 print"(down)programm ins basic. es wird jedoch nicht"
1170 print"geloescht (!)."
1180 print"(down)ein neustart erfolgt ueber den befehl"
1190 print"(down)(rvs)run(off)                    (rvs)<return>"
1200 goto530
```

Dabei handelt es sich um ein Diskettenverwaltungsprogramm wie Listing 7.1, ist aber mit einem komfortablen Menü versehen. Dabei wurde die Möglichkeit des Programmabbruchs entfernt, ansonsten ist die Steuerroutine in den Zeilen 300–460 parallel zu den Zeilen 300–470 in Listing 7.3 aufgebaut. Neu ist noch die Ausgabe von Infotexten zum aktuellen Menüpunkt über <CTRL C>.

7.1.5 Horizontales Menü

Sollen aus irgendwelchen Gründen die Menüpunkte horizontal angeordnet werden, ist auch dies möglich. Allerdings muß dann das Inversfeld links oder rechts bewegt werden, aber <CRSR DOWN> und <CRSR UP> wären dafür keine plausiblen Tasten. Am Algorithmus ändert sich nichts, aber die Funktion von <CRSR DOWN> wird bei horizontaler Anordnung von <CRSR RIGHT> übernommen, während <CRSR LEFT> die Taste <CRSR UP> ersetzt.

Listing 7.5: MENUE-BSP. 5

```
100 if peek(769)<>243 then print"erst paradoxon basic laden":stop
110 :
120 :
130 rem ********************************
140 rem *                              *
150 rem *  menue-beispielprogramm #5   *
160 rem *                              *
170 rem *  anwendungsbeispiel eines    *
180 rem *    horizontalen menues       *
190 rem *                              *
200 rem ********************************
210 rem *                              *
220 rem * written 12.10.1987 by        *
230 rem *                              *
240 rem *      florian mueller         *
250 rem *                              *
260 rem ********************************
270 :
280 :
290 :
300 clr:dim rt(4):p=0:rt(p)=1
310 poke53280,0:poke53281,0:print"(clr)"
320 print(0,3)"(cyn)menuesteuerung - beispielprogramm"
330 print(1,3)"--------------------------------------"
340 poke199,rt(0):print(9, 0)"(red)pkt. a(off)"
350 poke199,rt(1):print(9, 9)"(grn)pkt. b(off)"
360 poke199,rt(2):print(9,18)"(lblu)pkt. c(off)"
370 poke199,rt(3):print(9,26)"(yel)pkt. d(off)"
380 poke199,rt(4):print(9,34)"(gry3)pkt. e(off)(wht)"
390 get t$:if t$="" then 390
400 if t$=chr$(13) then print"(down)(down)(down)angewaehlter menuepunkt: "chr$(6
5+p)" /"p:end
410 if t$=chr$(3) then print"(down)(down)programmabbruch.":end
420 rt(p)=0
430 if t$=chr$(29) then p=p+1+5*(p=4)
440 if t$=chr$(157)then p=p-1-5*(p=0)
450 if t$=chr$(19) then p=0
460 rt(p)=1
470 goto 320
```

Der größte Teil von Listing 7.5 wurde aus Listing 7.3 übernommen, doch die PRINT-Zeilen 340–380 wurden so geändert, daß nun eine horizontale Ausgabe erfolgt. Da alle Menüpunkte in einer Zeile stehen, ist auch die Zeilenangabe (Y-Koordinate bei PRINT) jeweils 9.

Außerdem wurden die IF-Abfragen in den Zeilen 430 und 440 so geändert, daß CHR$(29) für <CRSR RIGHT> anstelle von CHR$(17) für <CRSR DOWN> sowie CHR$(157) für <CRSR LEFT> anstelle von CHR$(145) für <CRSR UP> steht.
Die letzte tiefgreifende Änderung besteht darin, daß jetzt die Menüpunkte nicht mehr als Routinen über ON...GOTO aufgerufen werden, sondern – weil das Prinzip von ON...GOTO klar ist – nur die Nummer des angewählten Menüpunktes ausgegeben wird (Zeile 400).
Noch ein Wort zu Zeile 400, wo außer der Menüpunkt-Nummer auch die Buchstabenbezeichnung (A–E) ausgegeben wird: Der Code des Buchstaben »A« ist die 65 (siehe C64-Handbuch). Alle im Alphabet folgenden Buchstaben haben jeweils eine um 1 größere Codenummer. Aufgrund dieser Tatsache läßt sich der Buchstabe des Menüpunktes mit der kleinen Rechnung »CHR$(65+P)« ermitteln.

7.1.6 Zweidimensionales Menü

Ein zweidimensionales Menü bringen Sie jetzt bitte bei eingelegter Programmdiskette unter Paradoxon Basic mit LOAD"MENUE-BSP. 6" und RUN auf den Bildschirm.

Listing 7.6: MENUE-BSP. 6

```
100 if peek(769)<>243 then print"erst paradoxon basic laden":stop
110 :
120 :
130 rem ********************************
140 rem *                              *
150 rem *  menue-beispielprogramm #6   *
160 rem *                              *
170 rem *   zweidimensionales menue    *
180 rem *  mit 3 zeilen zu 2 spalten   *
190 rem *                              *
200 rem ********************************
210 rem *                              *
220 rem * written 13.10.1987 by        *
230 rem *                              *
240 rem *      florian mueller         *
250 rem *                              *
260 rem ********************************
270 :
280 :
290 :
300 clr:dim rt(5):p=0:rt(p)=1
310 poke53280,0:poke53281,0:print"(clr)"
320 print(0,3)"(cyn)menuesteuerung - beispielprogramm"
330 print(1,3)"----(zweidimensionales menue)----"
340 poke199,rt(0):print( 6, 9)"(red)punkt a"
350 poke199,rt(1):print( 8, 9)"(grn)punkt b"
360 poke199,rt(2):print(10, 9)"(lblu)punkt c"
370 poke199,rt(3):print( 6,19)"(yel)punkt d"
380 poke199,rt(4):print( 8,19)"(gry3)punkt e"
390 poke199,rt(5):print(10,19)"(lgrn)punkt f"
400 get t$:if t$="" then 400
410 if t$=chr$(13) then print"(down)(down)angewaehlter punkt: "chr$(65+p)" /"p:e
nd
420 if t$=chr$(3) then print"(down)(down)programmabbruch.":end
430 rt(p)=0
440 if t$=chr$(17) then p=p+1+3*(p=2 or p=5)
```

```
450 if t$=chr$(145)then p=p-1-3*(p=0 or p=3)
460 if t$=chr$(29) then p=p+3+6*(p>2)
470 if t$=chr$(157)then p=p-3-6*(p<3)
480 if t$=chr$(19) then p=0
490 rt(p)=1
500 goto 320
```

Im zweidimensionalen Menü werden alle vier Cursortasten miteinbezogen. <CRSR UP> und <CRSR DOWN> werden im Prinzip wie im vertikalen Menü zur Ansteuerung der vertikalen Ebenen (A–D, B–E, C–F) verwendet. <CRSR RIGHT> und <CRSR LEFT> schalten zwischen den horizontalen Ebenen (A-B-C, D-E-F) um. Die entsprechenden Berechnungen stehen in den Zeilen 440–480. Diese müssen nicht wieder aufgeschlüsselt werden, da wir dies schon bei »MENUE-BSP. 2« getan haben. Deshalb bringt Ihnen ein Flußdiagramm sicherlich größeren Nutzen (Bild 7.4):

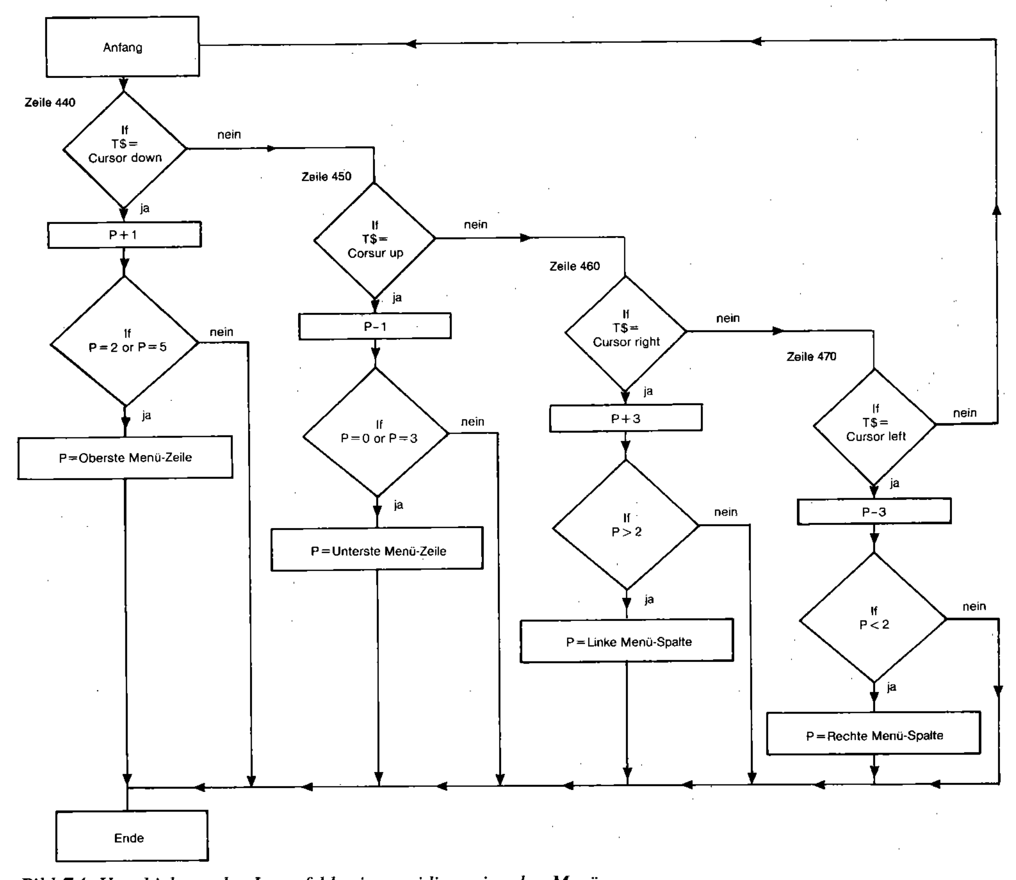


Bild 7.4: Verschiebung des Inversfeldes im zweidimensionalen Menü

7.1.7 Ja-/Nein-Abfragen mit horizontalem Menü

Fast kein Programm kommt ohne eine Abfrage aus, die mit »JA« oder »NEIN« zu beantworten ist. Natürlich kann diese Information über INPUT eingeholt werden, aber folgendes Programm hat eine bessere Lösung dieses Problems parat:

Listing 7.7: MENUE-BSP. 7

```
100 if peek(769)<>243 then print"erst paradoxon basic laden":stop
110 :
120 :
130 rem ********************************
140 rem *                              *
150 rem *  menue-beispielprogramm #7   *
160 rem *                              *
170 rem *   sicherheitsabfragen mit    *
180 rem * zwei auswahlmoeglichkeiten   *
190 rem *                              *
200 rem ********************************
210 rem *                              *
220 rem * written 13.10.1987 by        *
230 rem *                              *
240 rem *       florian mueller        *
250 rem *                              *
260 rem ********************************
270 :
280 :
290 :
300 clr:p=0
310 poke53280,0:poke53281,0:print"(clr)"
320 print(0,3)"(cyn)menuesteuerung - beispielprogramm"
330 print(1,3)"----(sicherheitsabfrage j/n)-----"
340 print(5,0)"waehlen sie aus (ja oder nein):"
350 if p=0 then poke 199,1
360 print(5,32)"ja"
370 if p then poke 199,1
380 print(5,36)"nein"
390 get t$:if t$="" then 390
400 if t$=chr$(29) or t$=chr$(157) or t$=" " then p=(not p)and1:rem >>flippen<<
410 if t$="j" then p=0:goto 350
420 if t$="n" then p=1:goto 350
430 if t$=chr$(3) then print"(down)(down)abbruch.":end
440 if t$<>chr$(13) then 350
450 if p=0 then print"(down)(down)sie haben sich fuer >>ja<< entschieden."
460 if p=1 then print"(down)(down)ihre antwortet lautet >>nein<<."
```

Dabei wird mit <CRSR RIGHT>, <CRSR LEFT> oder der Leertaste zwischen »JA« und »NEIN« hin- und hergeschaltet. Da nur zwei Auswahlmöglichkeiten (Menüpunkte) vorhanden sind, gibt es in der Wirkung keinen Unterschied zwischen Links- und Rechts-Bewegung des Inversfeldes. Deshalb werden alle drei Tasten zu diesem Zweck mit einer einzigen Zeile (400) behandelt; deren THEN-Teil ändert den Wert P von 0 auf 1 und von 1 auf 0. Dieses Wechseln zwischen zwei Zuständen nennt man »flippen« (umdrehen). Es wird durch NOT und AND realisiert:

... p = (not p) and 1

Falls Sie einmal ein solches Flippen programmieren müssen, so greifen Sie doch auf diese Formel zurück. Zwischen –1 und 0 bzw. 0 und –1 wird übrigens durch »P=NOT P« gewechselt. Bemerkenswert an Listing 7.7 sind noch die Zeilen 350 und 370, welche in diesem Fall das Anlegen eines Arrays RT() überflüssig machen: Bei so wenigen Ausgabezeilen ist es durchaus vertretbar, vor jeder der beiden Bildschirmausgaben mit IF...THEN die Variable P zu prüfen. Der Ausdruck »IF P THEN« ist dabei in Zeile 370 nur eine Kurzschreibweise für »IF P< >0 THEN«, womit wiederum letztlich »IF P=1 THEN« gemeint ist.

7.1.8 Universelle Routine zur komfortablen Menüsteuerung

Die Programmierung der komfortablen Menüs ist zwar reizvoll, aber zugleich auch mit einem gewissen Aufwand verbunden. Deshalb bietet es sich in Programmen mit mehreren Menüs an, eine flexible Unterroutine einzusetzen:

Listing 7.8: MENUE-BSP. 8

```
100 if peek(769)<>243 then print"erst paradoxon basic laden":stop
110 :
120 :
130 rem ******************************
140 rem *                            *
150 rem *  menue-beispielprogramm #8  *
160 rem *                            *
170 rem *     die endgueltige und    *
180 rem *  universelle menue-routine  *
190 rem *                            *
200 rem ******************************
210 rem *                            *
220 rem * written 13.10.1987 by      *
230 rem *                            *
240 rem *       florian mueller      *
250 rem *                            *
260 rem ******************************
270 :
280 :
290 :
300 rem >>>demonstrationsteil<<
310 :
320 am=4: rem *** anzahl der menuepunkte
330 fi=-1:rem *** initialisierung >>ja<<
340 for i=0 to am-1
350 :mp(0,i) = 11*i:rem *** textspalte
360 :mp(1,i) = 20:  rem *** textzeile
370 :mp$(i)  = "punkt "+chr$(65+i):rem *** menuepunkt-text
380 next
390 print"(clr)"
400 ab$=chr$(3):rem abbruchtaste
410 hv=-1:    rem flag fuer >>horizontal<<
420 print"bitte auswaehlen:"
430 gosub 500
440 print"(down)(down)angewaehlter menuepunkt:"ap
450 end
460 :
470 :
480 :
```

```
490 :
500 rem ******************************
510 rem *                            *
520 rem *  universelle menueroutine  *
530 rem *                            *
540 rem *  als ablauffertiges modul  *
550 rem *                            *
560 rem ******************************
570 :
580 :
590 iffithenap=0:fori=0toam-4:rf(i)=0:next:rf(ap)=1
600 fori=0toam-1:poke199,rf(i):print(mp(1,i),mp(0,i))mp$(i):next
610 gett$:ift$=""then610
620 ift$=chr$(13)ort$=ab$thenreturn
630 rf(ap)=0
640 if(t$=chr$(17)andnothv)or(t$=chr$(29)andhv)thenap=ap+1+am*(ap=(am-1))
650 if(t$=chr$(145)andnothv)or(t$=chr$(157)andhv)thenap=ap-1-am*(ap=0)
660 ift$=chr$(19)thenap=0
670 rf(ap)=1
680 goto600
```

Dieses Programm besteht aus einem Demonstrationsteil (Zeilen 300–450) und der eigentlichen Unterroutine (Zeilen 500–680). Die Menüroutine wird über GOSUB 500 aufgerufen und benötigt folgende Parameter:

Anzahl der Menüpunkte: Variable AM

In AM muß die Anzahl der Menüpunkte stehen. Dabei darf man sich nicht davon beirren lassen, daß die Menüpunkte intern mit 0 beginnend numeriert werden. Zwei Menüpunkte ist die niedrigste sinnvolle Anzahl, nach oben besteht eigentlich keine Grenze außer der Bildschirmkapazität, da die Menüpunkte schließlich auch am Bildschirm durch einen Text vertreten sein wollen.

Flag für Initialisierung: Variable FI

Wenn FI den Wert -1 hat, wird das Menü intialisiert, d.h., das Inversfeld wird auf den obersten Menüpunkt gesetzt. Beim ersten Aufruf eines Menüs ist dies dringend anzuraten. Andernfalls ist FI mit dem Wert 0 zu belegen.

Position eines Menüpunktes am Bildschirm: Array MP(,)

Die Spalte eines Menüpunktes »n« am Bildschirm muß in MP(0,n) stehen, die Zeile in MP(1,n). Die Menüpunkte werden, wie bisher auch, mit 0 beginnend numeriert. Es liegt in der Verantwortung der aufrufenden Programme, daß Zeile und Spalte sinnvoll gewählt sind (Menüpunkte mit höherer Nummer müssen weiter unten bzw. weiter rechts stehen).

Text eines Menüpunktes am Bildschirm: Array MP$()

Der Text, der am Bildschirm für einen Menüpunkt »n« steht, ist in MP$(n) abzulegen. Dieser String darf auch Steuerzeichen, insbesondere zur Farbeinstellung, beinhalten; doch bei Verwendung von <RVS OFF> (ASCII-Code: 146) gefährdet man die korrekte Invertierung des Textes.

Taste zum Verlassen des Menüs: Variable AB$

Außer <RETURN> läßt sich in AB$ eine weitere Taste definieren, bei deren Auslösen das Menü-Unterprogramm zurückkehrt. Das Beispielprogramm läßt beispielsweise <CTRL C> zu (Zeile 400). Wenn ein Programm viele hierarchische Menüs hat (Ober- und Unter-Menüs, gegebenenfalls noch Untermenüs zweiten und dritten Grades), sollten Tasten das Anspringen des Hauptmenüs sowie des unmittelbar in der Hierarchie darüber- oder darunterliegenden Menüs ermöglichen. Eine Taste dazu läßt sich in AB$ ablegen; für die Definition mehrerer Tasten ist dann die Routine in Zeile 620 zu modifizieren.
Nach Verlassen des Menüs kann das Hauptprogramm anhand von T$ feststellen, welche Taste das Verlassen des Menüs bewirkt hatte.

Horizontal-/Vertikal-Flag: Variable HV

HV=0: Vertikales Menü, in welchem die Tasten <CRSR DOWN> und <CRSR UP> das Inversfeld bewegen.
HV=-1: Horizontales Menü, gesteuert über <CRSR RIGHT> und <CRSR LEFT>.

Zweidimensionale Menüs sind nicht vorgesehen, da diese niemals in derartiger Häufung auftauchen, daß eine universelle Routine lohnend ist. Ganz abgesehen davon würde auch die Arbeitsgeschwindigkeit sehr darunter leiden.

Nummer des angewählten Menüpunktes: Variable AP

In AP steht nach dem GOSUB-Aufruf der Routine die Nummer des angewählten Menüpunktes. Wenn AP an die Routine übergeben werden soll, um die Position des Inversfeldes festzulegen, muß das Initialisierungsflag FI mit 0 belegt sein; außerdem muß das Feld RF(), welches die Revers-Informationen enthält, entsprechend gesetzt werden, was bei FI = –1 der Initialisierungsteil der Unterroutine übernimmt.

Variablendimensionierungen

Wenn viele Menüpunkte existieren, müssen auch viele Informationen in den Arrays MP(,), MP$() und RF() – letzteres wird von der Unterroutine als Hilfsarray verwendet – stehen. Da die automatische Variablendimensionierung seitens des Basic-Interpreters nur bis zu 10 Elemente umfaßt, sind die genannten Arrays bei mehreren Menüpunkten wie folgt zu dimensionieren:

```
DIM MP(1,n-1),MP$(n-1),RF(n-1)
```

»n« ist dabei die konstante Anzahl der Menüpunkte. Für die Dimensionierung ist das Hauptprogramm selbst verantwortlich.

7.1.9 Farbeinstellung über Menü

»Allen Menschen recht getan, ist eine Kunst, die keiner kann.« – So oder ähnlich könnte man das Problem eines Programmierers beschreiben, der die Bildschirmfarben seines Werkes festlegen muß. Denn der eine Anwender hat diesen Monitor, der andere jenen, und so kommt es vor, daß der eine die grüne Schrift auf schwarzem Grund am besten lesen kann, der andere aber diese Farben partout nicht mag.

Abhilfe schafft hier nur die individuelle Einstellmöglichkeit über eine Farbwahlroutine, die jedem Anwender seine eigene Farbkombination gestattet, damit (hoffentlich) jeder zufrieden ist. Dabei stehen Zeichen-, Rahmen- und Hintergrundfarbe zur Wahl.

Listing 7.9: MENUE-BSP. 9

```
100 if peek(769)<>243 then print"erst paradoxon basic laden":stop
110 :
120 :
130 rem ********************************
140 rem *                              *
150 rem *  menue-beispielprogramm #9   *
160 rem *                              *
170 rem *   farbeinstellung ueber      *
180 rem *    komfortables menue        *
190 rem *                              *
200 rem ********************************
210 rem *                              *
220 rem * written 13.10.1987 by        *
230 rem *                              *
240 rem *       florian mueller        *
250 rem *                              *
260 rem ********************************
270 :
280 :
290 :
300 clr:dimrt(2):ap=0:rt(ap)=1:h=15:r=6:v=0:print"(clr)"
310 poke53280,r:poke 53281,h:poke 646,v
320 print"(home)bitte stellen sie nun ihre persoenliche farbkombination ein !"
330 poke199,rt(0):print(10,2)"hintergrundfarbe aendern (+ oder -)"
340 poke199,rt(1):print(12,5)"rahmenfarbe aendern (+ oder -)"
350 poke199,rt(2):print(14,2)"vordergrundfarbe aendern (+ oder -)"
360 gett$:ift$=""then360
370 ift$=chr$(17)thenrt(ap)=0:ap=ap+1+3*(ap=2):rt(ap)=1:goto310
380 ift$=chr$(145)thenrt(ap)=0:ap=ap-1-3*(ap=0):rt(ap)=1:goto310
390 ift$=chr$(19)thenrt(ap)=0:ap=0:rt(0)=1:goto310
400 ift$<>"+"then450
410 :ifap=0thenh=(h+1)and15:ifh=vthen410
420 :ifap=1thenr=(r+1)and15
430 :ifap=2thenv=(v+1)and15:ifv=hthen430
440 :goto310
450 ift$<>"-"then500
460 :ifap=0thenh=(h-1)and15:ifh=vthen460
470 :ifap=1thenr=(r-1)and15
480 :ifap=2thenv=(v-1)and15:ifv=hthen480
490 :goto310
500 ift$<>chr$(13)then360
```

Das Inversfeld wird wie in anderen Menüs dieser Art bewegt, durch »+« oder »–« schaltet man die jeweils invers angezeigte Farbe weiter. Diese Änderungen werden sofort sichtbar, und durch <RETURN> beendet man die Farbeinstellung, wobei ab Zeile 500 das eigentliche Hauptprogramm anzuhängen ist. Dieses kann entweder über PEEK-Funktionen oder aus den Variablen H, R und V die eingestellten Farben entnehmen:

H	Hintergrund	(Adresse 53281)
R	Rahmen	(Adresse 53280)
V	Vordergrund	(Adresse 646)

Das Einstellungsmodul achtet übrigens genauestens darauf, daß durch die Farbkombinationen kein »unsichtbarer« Bildschirm zustandekommt, weil Hintergrund- und Vordergrund-Farbe übereinstimmen. Vielleicht kennen Sie den Witz von der ostfriesischen Flagge? – Weißer Adler auf weißem Grund!

7.1.10 Es geht auch ohne Paradoxon Basic!

Der Einsatz von Paradoxon Basic ist zwar sehr empfehlenswert, doch in folgenden zwei Fällen störend:

1. Programme, die sich auf Paradoxon Basic stützen, lassen sich nicht kompilieren.
2. Die Weitergabe von Programmen erzwingt die gleichzeitige Weiterverbreitung von Paradoxon Basic.

Deshalb sei am Beispiel der Farbwahlroutine gezeigt, wie man auch ohne Paradoxon Basic die gesamten Menütechniken anwendet:

Listing 7.10: MENUE-BSP. 10

```
100 rem *** kein paradoxon-test ***
110 :
120 :
130 rem ********************************
140 rem *                              *
150 rem *  menue-beispielprogramm #10  *
160 rem *                              *
170 rem *    farbeinstellung ohne      *
180 rem *       paradoxon basic        *
190 rem *                              *
200 rem ********************************
210 rem *                              *
220 rem * written 13.10.1987 by        *
230 rem *                              *
240 rem *      florian mueller         *
250 rem *                              *
260 rem ********************************
270 :
280 :
290 :
300 clr:dimrt(2):ap=0:rt(ap)=1:h=15:r=6:v=0:print"(clr)"
310 poke53280,r:poke 53281,h:poke 646,v
320 print"(home)bitte stellen sie nun ihre persoenliche farbkombination ein !"
```

```
330 poke199,rt(0):z=10:s=2:gosub1000:print"hintergrundfarbe aendern (+ oder -)"
340 poke199,rt(1):z=12:s=5:gosub1000:print"rahmenfarbe aendern (+ oder -)"
350 poke199,rt(2):z=14:s=2:gosub1000:print"vordergrundfarbe aendern (+ oder -)"
360 gett$:ift$=""then360
370 ift$=chr$(17)thenrt(ap)=0:ap=ap+1+3*(ap=2):rt(ap)=1:goto310
380 ift$=chr$(145)thenrt(ap)=0:ap=ap-1-3*(ap=0):rt(ap)=1:goto310
390 ift$=chr$(19)thenrt(ap)=0:ap=0:rt(0)=1:goto310
400 ift$<>"+"then450
410 :ifap=0thenh=(h+1)and15:ifh=vthen410
420 :ifap=1thenr=(r+1)and15
430 :ifap=2thenv=(v+1)and15:ifv=hthen430
440 :goto310
450 ift$<>"-"then500
460 :ifap=0thenh=(h-1)and15:ifh=vthen460
470 :ifap=1thenr=(r-1)and15
480 :ifap=2thenv=(v-1)and15:ifv=hthen480
490 :goto310
500 ift$<>chr$(13)then360
1000 poke781,z:poke782,s:sys65520:return
```

Da aus der Befehlserweiterung von Paradoxon Basic nur der erweiterte PRINT-Befehl Verwendung findet, läßt sich die Koordinatenangabe auch durch die Befehlsfolge POKE781,zeile:POKE782,spalte:SYS65520 ersetzen, wodurch ebenfalls eine Cursorpositionierung erfolgt.
Im Beispielprogramm ist dies als Unterprogramm ab Zeile 1000 abgelegt. Da dieses »versehentlich« bei Drücken von <RETURN> ausgeführt wird, tritt ein RETURN WITHOUT GOSUB ERROR auf, der bei Einfügen eigener Programmteile ab Zeile 501 verschwindet.

7.1.11 Mega-Menüs

Mit dem Beispielprogramm »MENUE-BSP. 11 BA« sei ein Extremfall der Menüsteuerung vorgeführt: Sage und schreibe 15 Menüpunkte, angordnet in 3 Spalten zu je 5 Zeilen, zaubert dieses Programm auf den Bildschirm.

Listing 7.11: MENUE-BSP. 11 BA

```
100 clr:dim rt(14):p=0:rt(p)=1
110 poke53280,0:poke53281,0:print"(clr)"
120 z=0:s=3:gosub1000:print"(cyn)menuesteuerung - beispielprogramm"
130 z=1:gosub1000:print"----(zweidimensionales menue)----"
140 poke199,rt(0):z=6:s=6:gosub1000:print"(red)punkt a"
150 poke199,rt(1):z=8:gosub1000:print"(grn)punkt b"
160 poke199,rt(2):z=10:gosub1000:print"(lblu)punkt c"
170 poke199,rt(3):z=12:gosub1000:print"(yel)punkt d"
180 poke199,rt(4):z=14:gosub1000:print"(gry3)punkt e"
190 poke199,rt(5):z=6:s=16:gosub1000:print"(lgrn)punkt f"
200 poke199,rt(6):z=8:gosub1000:print"(orng)punkt g"
210 poke199,rt(7):z=10:gosub1000:print"(brn)punkt h"
220 poke199,rt(8):z=12:gosub1000:print"(lred)punkt i"
230 poke199,rt(9):z=14:gosub1000:print"(gry2)punkt j"
240 poke199,rt(10):z=6:s=26:gosub1000:print"(red)punkt k"
250 poke199,rt(11):z=8:gosub1000:print"(pur)punkt l"
```

```
260 poke199,rt(12):z=10:gosub1000:print"(grn)punkt m"
270 poke199,rt(13):z=12:gosub1000:print"(cyn)punkt n"
280 poke199,rt(14):z=14:gosub1000:print"(blu)punkt o"
290 get t$:if t$="" then 290
300 if t$=chr$(13) then print"(down)(down)angewaehlter punkt: "chr$(65+p)" /"p:e
nd
310 if t$=chr$(3) then print"(down)(down)programmabbruch.":end
320 rt(p)=0
330 if t$=chr$(17) then p=p+1+5*(p=4 or p=9 or p=14)
340 if t$=chr$(145)then p=p-1-5*(p=0 or p=5 or p=10)
350 if t$=chr$(29) then p=p+5+15*(p>9)
360 if t$=chr$(157)then p=p-5-15*(p<5)
370 if t$=chr$(19) then p=0
380 rt(p)=1
390 goto 120
1000 poke781,z:poke782,s:sys65520:return
```

Da auch noch auf den Einsatz von Paradoxon Basic verzichtet wird (siehe Unterprogramm ab Zeile 1000, übernommen aus Listing 7.10), ergibt sich eine deutliche Verzögerung des Programmablaufs, welche – seien wir ehrlich – die Bedienung ins Unerträgliche zieht. Sicherlich könnte man nun die Optimierungstechniken aus Kapitel 6.1 in die Schlacht um Rechenzeit werfen, doch dies hat erfahrungsgemäß nur bis zu etwa acht Menüpunkten einen Sinn; bei 15 Menüpunkten ist selbst dies nicht ausreichend. Deshalb bleibt nur ein Ausweg, der ebenfalls in Kapitel 6 erwähnt wird: das Kompilieren des Programms (siehe 6.2).
Starten Sie nun die Datei »C/MENUE-BSP. 11«; Sie werden von der angenehmen Geschwindigkeit sicher erfreut sein. Dabei ist zu beachten, daß nur Standard-Basic-Programme kompilierbar sind; deshalb mußten wir ohne Paradoxon Basic auskommen.
Was ich Ihnen zeigen wollte, ist folgendes: Hinsichtlich der Geschwindigkeit sind auch die Menüs gewissen Grenzen unterworfen. Bei Compiler-Einsatz jedoch lösen sich viele Probleme dieser Art. Doch auch die Menüprogrammierung in Maschinensprache hat ihren Wert; in Kapitel 10 erfahren Sie u.a., wie man solche Menüs in Assembler realisiert. Sie erhalten dazu ein fertiges Grundgerüst für eigene Anwendungen, welches Ihnen über mögliche Anfangsschwierigkeiten schnell hinweghilft.

7.1.12 An AND denken!

Abschließend sei noch kurz ein sehr effektiver Trick erwähnt, wie man durch den Einsatz des Befehls AND die Menüprogrammierung in günstigen Ausnahmefällen sehr vereinfacht. Aus Platzgründen (die Vorderseite der Diskette ist nun ausgelastet, alle weiteren Kapitel beziehen sich auf die Rückseite) sei dafür nur die Technik erwähnt, Beispiele ergeben sich fast von selbst. Greifen wir noch einmal auf 7.1.7 zurück; dort wurde für den Sonderfall »2 Menüpunkte« durch den Einsatz des AND-Befehls die Berechnung erleichtert. Für 4 Menüpunkte, die natürlich von

0 bis 3 numeriert sind, wären folgende IF-Behandlungen für <CRSR UP> und <CRSR DOWN> erforderlich:

```
IF T$=CHR$(17) THEN P=P+1+4*(P=3)
IF T$=CHR$(145)THEN P=P-1-4*(P=0)
```

Sieht man die Parallelen, so schließt man daraus, daß jeweils der Wert von P um 1 verringert oder um 1 erhöht wird; anschließend erfolgt eine »Korrektur«, damit das Inversfeld nicht außerhalb des Bildschirms liegen soll, sondern irgendein Element von RT() durch P indiziert ist. Für die entsprechenden Zeilen ergibt sich jedoch noch ein sehr schneller und einfacher Lösungsweg:

```
IF T$=CHR$(17) THEN P=(P+1)AND3
IF T$=CHR$(145)THEN P=(P-1)AND3
```

Der Trick ist, daß »AND 3« automatisch Ergebnisse im Bereich von 0 bis 3 liefert; »–1« wird dabei zu »3«, »4« hingegen zu »0«. In binärer Darstellung läßt sich dies nachweisen:

```
-1 AND 3 = %11111111 AND %00000011 = %000000011 = 3
 4 AND 3 = %00000100 AND %00000011 = %000000000 = 0
```

Für alle anderen Werte (0, 1, 2 und 3) ergibt die Operation »AND 3« keine Veränderung. Beispiel: 2 AND 3 = %00000010 AND %00000011 = %000000010 = 2
Da auf binärer Ebene gerechnet wird, ist der AND-Trick jedoch nur auf Zweierpotenzen anwendbar, d.h., die Anzahl der Menüpunkte muß eine Potenz von 2 sein. Dies gilt also AUSSCHLIESSLICH für 2 (siehe 7.1.7), 4 (unser Beispiel!), 8, 16, 32 usw.
Für die gültigen Beispiele läßt sich hingegen eine allgemeine »Formel« schaffen, wobei »A« die Anzahl der Menüpunkte (2, 4, 8, 16, 32 ...) ist, »P« wie gewohnt den Menüpunkt angibt und die Numerierung nach wie vor mit »0« beginnt:

```
(P+1) AND " A-1"  entspricht P+1+A*(P=" A—1" )
(P-1) AND " A-1"  entspricht P-1-A*(P=0)
```

Wenn Sie jetzt »MENUE-BSP. 4« zur Hand nehmen, welches glücklicherweise 4 (=2 im Quadrat) Menüpunkte hat, so dürfen Sie folgende Änderungen durchführen:

```
420 IF T$=CHR$(17) THEN P=(P+1)AND3
430 IF T$=CHR$(145)THEN P=(P-1)AND3
```

Da zweimal »AND3« vorkommt, wäre auch folgende Konstruktion denkbar:

```
420 IF T$=CHR$(17) THEN P=P+1
430 IF T$=CHR$(145)THEN P=P-1
435 P=P AND 3
```

Um nun einen krönenden Abschluß zu finden, sei noch für die absoluten Freaks eine IF-freie Lösung für die Zeilen 420–440 vorgestellt:

```
420 P=P-(T$=CHR$(17))
430 P=P+(T$=CHR$(145))
435 P=P AND 3
440 P=-P*(T$<>CHR$(19))
```

Sie geben sicher zu, daß dies durch seine Vertracktheit eine gewisse Faszination besitzt. Besonders günstig: Da IF. . .THEN nicht mehr verwendet wird, ist auch die Zusammenfassung dieser vier Zeilen in eine einzige möglich, wobei man sogar noch den Inhalt von 450 und 460 darin verpacken könnte. Dadurch wäre ein Menü in wenigen Zeilen möglich!
Ich möchte Sie jedoch nicht mit den Grundlagen weiter strapazieren, denn diese haben Sie längst erfahren. Wenn Sie die Vorzeichen beachten (wahre Ausdrücke erhalten –1 zugewiesen), können Sie auch selbst diese Zeilen aufschlüsseln.

7.2 Menüs unter GEOS

Im Zusammenhang mit Menüs und deren Programmierung auf dem C 64 halte ich es für ausgesprochen wichtig, auch die neue Benutzeroberfläche GEOS zu erwähnen.

7.2.1 Anwendung der drei Menü-Typen

Sicherlich kennen Sie als Anwender bereits »die neue Welt für C 64/128«, wie es kurz und treffend in der Werbung heißt. Bild 7.5 zeigt ein sogenanntes »Pull-down-Menü«, Bild 7.6 ein »Startmenü«.

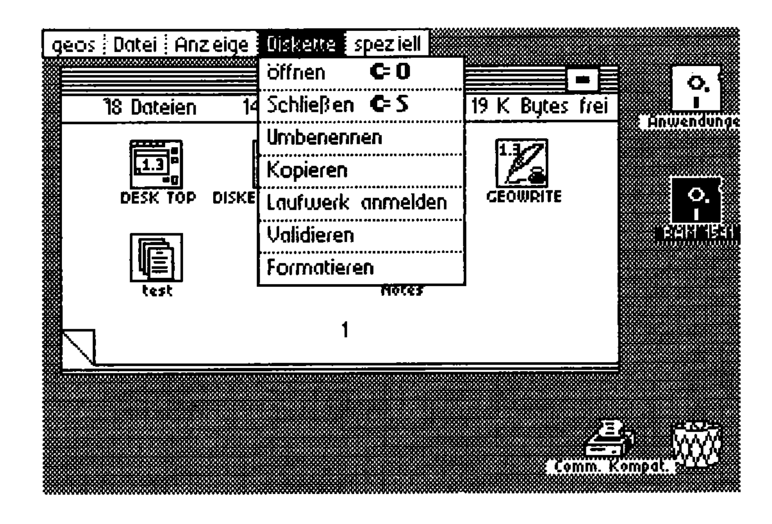


Bild 7.5: Pull-down-Menü unter GEOS

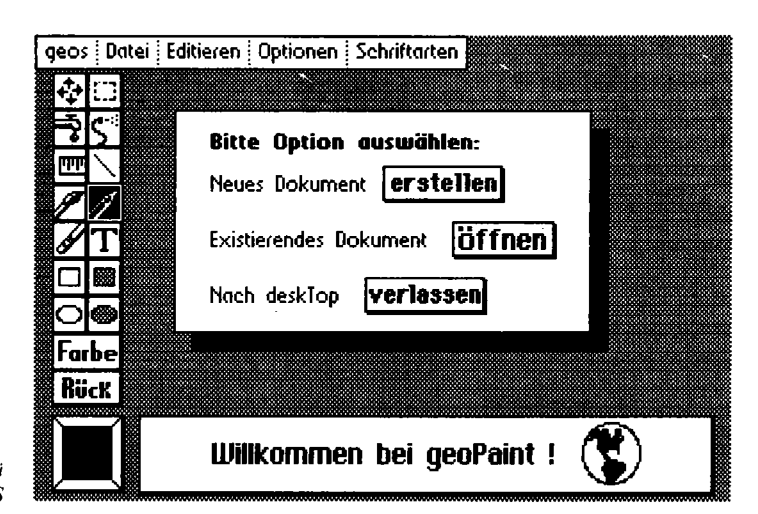


Bild 7.6: Startmenü unter GEOS

Ein Pull-down-Menü wird bedient, indem man einen der Punkte des Hauptmenüs anklickt; dies allein bewirkt jedoch oft keine Funktion, sondern hat zunächst zur Folge, daß ein weiteres Menü »abgerollt« wird (»pull down«). Im Beispiel wurde auf »Diskette« das vertikale Untermenü angeboten. Erst in diesen Menüs entscheidet man sich meist für endgültige Auswahlpunkte wie das »Öffnen« einer Diskette.
Nach Verwendung eines Pull-down-Menüs verschwindet es wieder spurlos vom Bildschirm und gibt den zuvor belegten Platz nicht nur frei, sondern stellt ihn auch exakt wieder her. Dies geschieht auch automatisch, wenn man das Menü durch Bewegung des Mauspfeils verläßt.
Eine weitere Besonderheit der Pull-down-Menüs liegt in der Möglichkeit, manche häufig benötigten Funktionen auch durch Tastendruck sofort zu erreichen (»Shortcuts« nennt man diese Tastenkombinationen). So ersetzt <C= O> das Anklicken von »Öffnen« im Menü »Diskette«.
Etwas anders sind Startboxen aufgebaut. In Bild 7.6 sehen Sie zwar links oben ein Pull-down-Menü, doch dieses ist zum dargestellten Zeitpunkt noch nicht aktiv; statt dessen muß der Anwender zwischen »erstellen«, »öffnen« und »verlassen« eine Entscheidung treffen; dies geschieht durch Bewegen des Mauspfeils in das entsprechende Befehlskästchen mit nachfolgendem Anklicken.
Eine dritte und letzte Form von Menü ist ebenfalls in Bild 7.6 sichtbar, aber noch nicht gültig: Links sehen Sie die sogenannte Werkzeugleiste des Zeichenprogramms GeoPaint, welche gewissermaßen ein grafisches Menü ist. Bis auf »Farbe« und »Rück« werden alle Optionen durch Kleingrafiken (Piktogramme) repräsentiert, was auch ungeübten Anwendern eine schnelle und problemlose Einarbeitung ermöglicht.

7.2.2 Zur Programmierung

Hier sei nur fragmentarisch erläutert, wie man unter GEOS die Menüsteuerung bewältigt. Eines ist dabei vorwegzunehmen: Das GEOS-Grundsystem bietet dazu zahlreiche Routinen an, welche die »Dreckarbeit« erledigen; der Programmierer eigener Applikationen muß nur dafür sorgen, daß zu einem Menü alle Daten bereitgestellt werden.
Deshalb enthält eine Menü-Tabelle folgende Spezifikationen:

- Position des Menüs am Bildschirm
- Größe des Menüs als Rechtecksangabe
- Anordnung: horizontal oder vertikal
- Darstellung: jederzeit sichtbar oder nur auf Abruf
- Texte der Menüpunkte
- Menütyp: Untermenü, Hauptmenü, verzweigendes Menü
- Adressen der Routinen zu einzelnen Menüpunkten
- Zeiger auf Tabellen der Untermenüs

Ist so eine Tabelle erst einmal angelegt, was nicht immer ganz leicht ist, so genügt der Aufruf der Routine MENU zur Aktivierung. Anschließend ist eine Warteschleife aufzubauen, die regelmäßig in die sogenannte GEOS-Hauptschleife springt; dort wird dann das Aktivieren eines Menüpunktes durch den Anwender erkannt und automatisch befolgt, d.h. die entsprechende Unterroutine wird angesprungen. Das Hauptprogramm selbst hat sich darum also nicht zu kümmern. Näheres über Anwendung und Programmierung erfahren Sie in »C64 – Alles über GEOS 1.2« bzw. »C64 – Alles über GEOS 1.3 (deutsch)«, wo alle Details zur Programmierung von Menüs unter GEOS aufgeführt sind, die Ihnen die Entwicklung eigener GEOS-Applikationen ermöglichen.

8 Windows oder: Fenster zum Bedienungskomfort

Der Begriff »Window« (Fenster) hat vor allem durch neuere Computer wie den AMIGA oder Atari ST einen größeren Bekanntheitsgrad erreicht, aber auch für den C 64 gibt es schon zahlreiche Programme, die eine sogenannte Window-Technik bieten. Die konsequenteste Window-Technik auf dem C64 findet man bei GEOS sowie der Dateiverwaltung MasterBase (Markt & Technik Verlag AG).

8.1 Windows auf einfache Weise

Wir werden uns jetzt ohne Zusatzprogramme selbständig mit der Window-Programmierung befassen, und Sie werden sehen, daß selbst Einsteiger mit den vorgestellten Programmiertechniken keine Schwierigkeiten haben.

8.1.1 Begriffserklärung

Zunächst wollen wir exakt definieren, was ein Window ist. Es handelt sich dabei um einen Bildschirmausschnitt, der als Teilbildschirm eingeblendet wird und separat, d.h. vom Hauptbildschirm unabhängig, behandelt wird. Ein Window stellt somit einen eigenständigen Funktionsbereich dar.
Die Ausgabe eines Windows geschieht durch Überschreiben eines Teils des Gesamtbildschirms durch den Window-Text. Dadurch, daß der alte Bildschirm nicht gelöscht, sondern nur teilweise überlagert wird, ist der vorherige Bildschirminhalt noch zu einem beträchtlichen Teil erkennbar. Werden viele solche Überlagerungen durchgeführt, entsteht allerdings ein relativ chaotisches Aussehen des Bildschirms, da von den einzelnen Windows nur noch Bruchstücke zu sehen sind. Grundvoraussetzung ist, daß das jeweils »aktuelle« Window in vollem Umfang eingeblendet ist.
In Bild 8.1 sehen Sie die schematische Darstellung eines Windows. Die Größe des gesamten Bildschirms setzt sich aus 40 Zeichen mal 25 Zeilen zusammen, also insgesamt 1000 Zeichen. Sie sehen sofort, was ein Window ist: Ein Teilbereich von der 23. bis zu 36. Spalte und von der 5. bis zur 12. Zeile wird als Window definiert und erhält eine bestimmte Funktion. Zum Beispiel

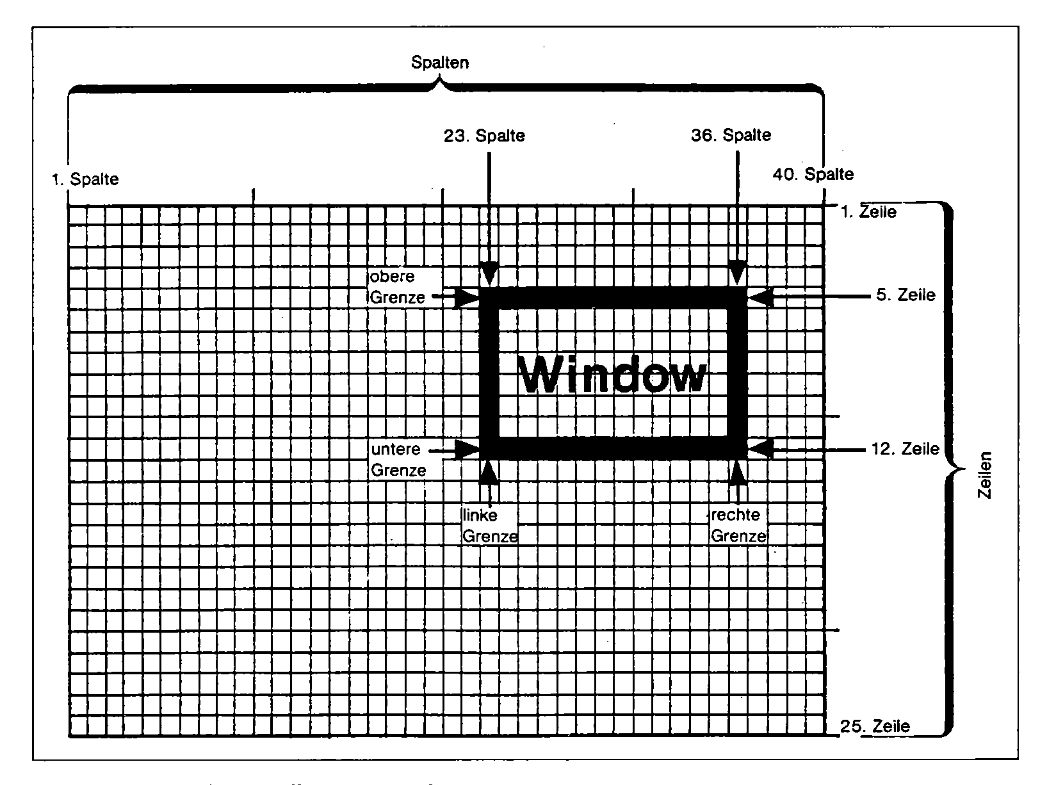


Bild 8.1: Schematische Darstellung eines Windows

könnten wir programmieren, daß dort dauernd die Uhrzeit eingeblendet wird, es könnte ein Auswahlmenü erscheinen oder eine Fehlermeldung stehen. Es gibt unzählige Anwendungsmöglichkeiten!

8.1.2 Das erste Window-Programm

Wenn wir fürs erste keine größeren Ansprüche stellen, als daß ein als Window definierter Bereich beschrieben wird, genügt uns folgendes Programm (es befindet sich, wie alle anderen Programme in diesem Kapitel, auf der Rückseite der Programmdiskette):

Listing 8.1: WINDOW-BSP. 1

```
100 rem ******************************
110 rem *                            *
120 rem *  anzeige einer information *
130 rem *                            *
140 rem *       in einem window      *
150 rem *                            *
160 rem ******************************
170 rem *                            *
180 rem * start mit zufaelligem be-  *
190 rem * schreiben des bildschirms  *
200 rem * zur hervorhebung des win-  *
210 rem * dows:'run' oder 'goto300'  *
220 rem *                            *
230 rem * start ohne aenderung des   *
240 rem * bildschirms: goto 360      *
250 rem *                            *
260 rem * c16,c64,c128(40-z.-modus)  *
270 rem *                            *
280 rem ******************************
290 :
300 rem bildschirm mit zufaelligen
310 rem zeichen beschreiben:
320 :
330 print chr$(147);:rem bildschirm loeschen
340 for f=1 to 950:print chr$(rnd(0)*48+48);:next f:rem 950 zufaellige zeichen
350 :
360 rem window ausgeben
370 rem ---------------
380 :
390 rem cursor in 5. zeile bewegen
400 :
410 print "(home)";:rem 'cursor home' ausfuehren
420 for f=1 to 4:print chr$(17);:next f:rem 4 zeilen ueberspringen
430 :
440 rem nun wird das window ausgegeben:
450 :
460 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)┌────────────┐(rg
ht)(rght)(rght)(rght)";
470 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)|>> window <<|(rg
ht)(rght)(rght)(rght)";
480 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)├────────────┤(rg
ht)(rght)(rght)(rght)";
490 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)|linke gr.:23|(rg
ht)(rght)(rght)(rght)";
500 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)|rechte g.:36|(rg
ht)(rght)(rght)(rght)";
510 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)|obere gr.: 5|(rg
ht)(rght)(rght)(rght)";
520 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)|untere g.:12|(rg
ht)(rght)(rght)(rght)";
530 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)└────────────┘";
540 :
550 get a$:if a$="" then 550
```

Wenn Sie es ganz normal über »RUN« starten, wird zunächst der Bildschirm mit zufälligen Zeichen aufgefüllt. Dies erleichtert die Demonstration, daß nur der Bildschirmbereich des Windows genutzt wird. Dann wird das Window ausgegeben; mit einem Tastendruck beendet man das Programm.
In diesem Anwendungsbeispiel wird das Window rein demonstrativ genutzt, es zeigt lediglich seine »persönlichen« Daten an. Rein von der Betrachtung des laufenden Programms kann man auch sehen, daß das Window optisch durch eine Umrahmung hervorgehoben wird. Diese Eingrenzung ist bei Commodore-Computern dank der Grafikzeichen einfach realisierbar, denn Kasten- und Rahmen-Symbole sind in ausreichender Menge vorhanden. Die grafische Hervorhebung ist von elementarer Bedeutung, damit man klar ersehen kann, wo das Window beginnt und wo es endet; Windows sind nämlich keine Spielerei, sondern sollen vielmehr eine ökonomische Nutzung des Bildschirms ermöglichen.
Bei einem Start des Programms über GOTO 360 wird das Überschreiben des Bildschirms durch zufällige Zeichen übersprungen. Sie könnten zum Beispiel das Programm über LIST ausgeben und dann ohne vorheriges Löschen des Bildschirms mit GOTO 360 starten. In jedem Fall wird das Window eingeblendet, egal was vorher auf dem Bildschirm stand. Genau diese Eigenschaft macht die Flexibilität der Windows aus.

8.1.3 Die Programmiertechnik

Bis jetzt haben wir uns sehr wenig mit der Technik beschäftigt und das Ganze eher aus Anwendersicht betrachtet. Sie werden aber sehen, daß man keine besonderen Programmierkenntnisse braucht, um Windows zu realisieren. Im Prinzip ist es nur eine Frage der Planung des Programms und im besonderen der Bildschirmausgabe. Bei neueren Computern wird eine ausgereifte Window-Technik bereits vom Betriebssystem unterstützt; deswegen brauchen wir aber noch lange nicht zu verzweifeln, denn vieles, was die Großen können, schaffen die Kleinen auch. GEOS ist der »lebende Beweis« dafür!
Kommen wir zurück zur Programmierung unter dem herkömmlichen C64-Betriebssystem, insbesondere in Basic 2.0. Weil die Ausgabe-Routinen des C64 nicht in der Lage sind, für uns darauf zu achten, daß wir nicht über die Grenzen eines Windows schreiben, haben wir solche Vorkehrungen selbst zu treffen. Als erstes müssen wir zur Ausgabe eines Windows auf die richtige Positionierung des Cursors achten. In unserem Beispiel (wir befinden uns immer noch bei Listing 8.1) muß der Cursor in die 5. Zeile bewegt werden, da der obere Rand des Windows mit 5 festgelegt ist. Dieses Ansteuern der 5. Bildschirmzeile erreichen wir in den Programmzeilen 410/420. Der Code CHR$(17) in Zeile 420 steht für <CRSR DOWN>.
Dadurch, daß in Zeile 420 insgesamt 4mal der Cursor von der obersten Zeile aus nach unten bewegt wird, landen wir logischerweise in der 5. Zeile. Falls es Ihnen nicht auf Anhieb klar sein sollte, so probieren Sie es doch im Basic-Editor mit den Tasten <HOME> und <CURSOR DOWN> aus.

Damit hätten wir schon die obere Grenze eingehalten. Von gleicher Wichtigkeit ist die linke Grenze. Wir können nicht ganz links am Bildschirmrand mit der Ausgabe einer Zeile beginnen, sondern erst in der 23. Spalte. Dies heißt für uns, daß wir die ersten 22 Spalten einer Bildschirmzeile überspringen müssen, wofür die CRSR-RIGHT-Bewegung verantwortlich ist (siehe Zeilen 460–530, wo es nur so von »(rght)« wimmelt).
Wenn wir dann eine richtige Position am Bildschirm angesteuert haben, kann eine Zeile des Windows ausgegeben werden. Es ist unbedingt darauf zu achten, daß der Text nicht länger wird, als das Window breit ist. Die Breite des Windows ergibt sich aus linkem und rechtem Rand. In unserem Fall ist der linke Rand die 23., der rechte Rand die 36. Spalte; die Breite beträgt 36–23+1 = 14 Zeichen. Die 1 muß deshalb addiert werden, da auch die Endspalten 23 und 36 genutzt werden. An den Zeilen 460–530 können Sie sehen, daß nach den CRSR-RIGHT-Steuerzeichen genau 14 Zeichen ausgegeben werden – nicht mehr und nicht weniger.
Am Ende der Zeile steht hinter den Anführungszeichen ein Semikolon, welches einen Zeilenvorschub verhindert. Wenn wir keine Steuerzeichen außer <CRSR RIGHT> verwenden, hat jetzt ein Ausgabestring in den Zeilen 460–530 genau 40 Zeichen Länge. In 530, der letzten Ausgabezeile, entfallen die vier CRSR-RIGHT-Codes am Schluß, da nach Ausgabe des Windows nicht wieder zum linken Rand zurückzukehren ist.
Der Strichpunkt ist aber weiterhin erforderlich, denn ein Zeilenvorschub kann den Bildschirmaufbau sehr durcheinanderbringen, wenn der Computer beispielsweise ein Scrolling unternimmt. Dieses Phänomen können Sie beobachten, wenn Sie das Programm per Tastendruck beenden und den Cursor nach unten bewegen.

8.1.4 Mehrere Windows gleichzeitig

Mit der vorgestellten Technik ist auch die gleichzeitige Darstellung mehrerer Fenster kein Problem mehr. Das Programm »WINDOW-BSP. 2« stellt ein Menü dar, wie es vor einem fiktiven Spielprogramm stehen könnte.

Listing 8.2: WINDOW-BSP. 2

```
100 rem ********************************
110 rem *                              *
120 rem * b e i s p i e l - m e n u e  *
130 rem *                              *
140 rem *      mit window-technik      *
150 rem *                              *
160 rem *  c16,c64,c128(40-z.-modus)   *
170 rem *                              *
180 rem ********************************
190 :
200 l=1:g=1:a=1:rem level, geschwindigkeit und anzahl der spieler vorbelegen
210 print chr$(147);
220 printtab(7)"beispielmenue fuer windows"
230 printtab(7)"--------------------------"
240 print"(down)(down)(down)(down)(down)"
250 print tab(3);chr$(18)" 1 "chr$(146);tab(10);"spielstufe(down)"
260 print tab(3);chr$(18)" 2 "chr$(146);tab(10);"geschwindigkeit(down)"
270 print tab(3);chr$(18)" 3 "chr$(146);tab(10);"anzahl der spieler(down)"
```

```
280 print tab(3);chr$(18)" 4 "chr$(146);tab(10);"auswahl beenden(down)(down)"
290 rem nach den normalen print-zeilen  werden jetzt die windows gedruckt
300 :
310 rem zuerst das copyright-window:
320 rem ===========================
330 :
340 print"(home)(down)(down)";
350 print"(rght)(rght)(rght)(rght)(rght)U─────────I(rght)(rght)(rght)(rght)(rght
)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(
rght)(rght)(rght)(rght)(rght)(rght)";
360 print"(rght)(rght)(rght)(rght)(rght)|copyright|(rght)(rght)(rght)(rght)(rght
)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(
rght)(rght)(rght)(rght)(rght)(rght)";
370 print"(rght)(rght)(rght)(rght)(rght)| window: |(rght)(rght)(rght)(rght)(rght
)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(
rght)(rght)(rght)(rght)(rght)(rght)";
380 print"(rght)(rght)(rght)(rght)(rght)├─────────┤(rght)(rght)(rght)(rght)(rght
)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(
rght)(rght)(rght)(rght)(rght)(rght)";
390 print"(rght)(rght)(rght)(rght)(rght)|(c) 64'er|(rght)(rght)(rght)(rght)(rght
)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(
rght)(rght)(rght)(rght)(rght)(rght)";
400 print"(rght)(rght)(rght)(rght)(rght)J─────────K"
410 :
420 rem nun das level-window:
430 rem ====================
440 print"(home)(down)(down)(down)(down)(down)";
450 print"(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rgh
t)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rvs)
  (rght)(rght)(rght)";
460 print"(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rgh
t)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rvs) spielstufe:"
+str$(l)+" (rght)(rght)(rght)";
470 print"(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rgh
t)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rvs)
  (off)"
480 :
490 rem und das geschwindigkeitswindow:
500 rem ==============================
510 print"(home)(down)(down)(down)(down)(down)(down)(down)(down)(down)(down)";
520 print"(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rgh
t)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)
(rght)(rght)(rght)┌──────────┐(rght)";
530 print"(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rgh
t)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)
(rght)(rght)(rght)|tempo:"+str$(g)+" |(rght)";
540 print"(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rgh
t)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)
(rght)(rght)(rght)└──────────┘"
550 :
560 rem und das spielerzahl-window:
570 rem ==========================
580 print"(home)(down)(down)(down)(down)(down)(down)(down)(down)(down)(down)(dow
n)(down)(down)(down)(down)(down)";
590 print"(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rgh
t)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)
*************(rght)(rght)";
600 print"(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rgh
t)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)
*"+str$(a)+" spieler *(rght)(rght)";
610 print"(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rgh
t)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)
*************(down)"+chr$(13)
620 :
630 :
640 print"bitte entsprechende taste druecken !"
650 :
```

```
660 get a$:if a$="" then 660
670 if a$ = "1" then l=l+1:if l>3 then l=1
680 if a$ = "2" then g=g+1:if g>3 then g=1
690 if a$ = "3" then a=a+1:if a>2 then a=1
700 if a$ = "4" then end
710 :
720 goto 290
```

Die Cursorprogrammierung wird dabei durch Verwendung von <CRSR DOWN> im Quote Mode (Zeilen 340, 440, 510 und 580) erleichtert. **Zur Bedienung:** Durch Drücken der Tasten <1> bis <3> wird der Wert für Geschwindigkeit, Anzahl der Spieler oder Spielstufe erhöht; wird der höchste erlaubte Wert überschritten, so wird wieder 1 eingestellt; <4> beendet das Programm.
An Listing 8.2 kann man auch verschiedene Möglichkeiten der optischen Eingrenzung eines Windows erkennen: Kasten- und Rahmensymbole, Reversschrift, Sternchen.
Bei mehreren Windows muß zusätzlich darauf geachtet werden, daß sich die Windows nicht gegenseitig ihren Platz wegnehmen, was manchmal viel Tüftelei verursacht.
Der Programmaufbau erklärt sich ansonsten durch die vielen Kommentare in REM-Zeilen von selbst.

8.1.5 Text unter Window retten

Einen großen Nachteil haben alle bisher von uns ausgegebenen Windows gehabt: Der Text, der vorher an der Position des Windows stand, ist verlorengegangen, weil er durch das Window überschrieben wurde. Dies macht in 50 Prozent aller Fälle nichts aus, aber was ist mit den anderen 50 Prozent?
Wir wollen den unter dem Window liegenden Bildschirmausschnitt retten, indem wir ihn über PEEK auslesen und in einem Variablenarray speichern. Soll dann das Window verschwinden und wieder der alte Text erscheinen, werden die Array-Inhalte wieder in den Bildschirmspeicher geschrieben.
Diese Anforderung erfüllt folgendes Programm, das Sie bitte zuerst ausprobieren (siehe Seite 270):

Listing 8.3: WINDOW-BSP. 3

```
100 rem ********************************
110 rem *                              *
120 rem *     anzeige eines windows    *
130 rem *                              *
140 rem * unter beruecksichtigung des *
150 rem *                              *
160 rem * unter dem window liegenden   *
170 rem *                              *
180 rem *      bildschirminhaltes      *
190 rem *                              *
200 rem ********************************
210 rem *                              *
220 rem *  c64 und c128 (40 zeichen)   *
230 rem *                              *
240 rem ********************************
250 :
260 b = 1024:rem basisadresse des bildschirmspeichers
270 dim t(111):rem array zum retten des textes (s. 350-)
280 :
290 get a$:if a$="" then 290
300 if a$="n" then 350:rem bei "n" bildschirm nicht zufaellig beschreiben
310 :
320 print chr$(147);
330 for f=1 to 999:print chr$(35+80*rnd(0));:next
340 :
350 rem an window-positionen liegenden
360 rem text retten
370 :
380 rem das window beansprucht:
390 rem die spalten 23-36
400 rem i.d. zeilen  5-12
410 :
420 i=0:rem mit index 0 beginnen
430 for z= 4 to 11:rem zeilen : 5-12
440 for s=22 to 35:rem spalten:23-36
450 t(i)=peek(b+40*z+s):i=i+1:rem wert einlesen, index erhoehen
460 next s,z
470 :
480 rem nun wird das window ausgegeben:
490 :
500 print"(home)(down)(down)(down)(down)";
510 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)┌────────────┐(rg
ht)(rght)(rght)(rght)";
520 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)|>> window <<|(rg
ht)(rght)(rght)(rght)";
530 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)├────────────┤(rg
ht)(rght)(rght)(rght)";
540 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)|linke gr.:23|(rg
ht)(rght)(rght)(rght)";
550 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)|rechte g.:36|(rg
ht)(rght)(rght)(rght)";
560 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)|obere gr.: 5|(rg
ht)(rght)(rght)(rght)";
570 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)|untere g.:12|(rg
ht)(rght)(rght)(rght)";
```

```
580 print "(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rg
ht)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght)(rght) └──────────┘";
590 :
600 get a$:if a$="" then 600
610 :
620 rem nun wird der text unter dem
630 rem window wieder angezeigt:
640 :
650 i=0
660 for z= 4 to 11
670 for s=22 to 35
680 poke b+40*z+s,t(i):i=i+1:rem wert schreiben, index erhoehen
690 next s,z
700 :
710 get a$:if a$="" then 710
720 :
730 goto 350:rem wieder retten und window anzeigen
```

Das Programm überschreibt auf Drücken einer Taste zunächst den Bildschirm mit zufälligen Zeichen, sofern nicht <N> gedrückt wurde, um den Bildschirm zu erhalten. Dann wird ein Window ausgegeben und auf einen Tastendruck gewartet, auf den hin der Text, welcher ursprünglich unter dem Window lag, wieder eingeblendet wird. Erneuter Tastendruck gibt wieder das Window aus, und dies können Sie beliebig oft wiederholen.

Das Window selbst kennen Sie bereits aus Listing 8.1. In Zeile 260 wird die Basisadresse des Bildschirmspeichers festgelegt (B=1024). Zeile 270 dimensioniert ein 111+1 = 112 Elemente großes Array T(). Das »+1« ergibt sich daraus, daß die Zählung mit 0 beginnt; zwischen 0 und 111 liegen 112 Elemente T(0)...T(111).

In diesem Array werden später die »unter dem Window liegenden« Zeichen gespeichert. Die Größe des Arrays ergibt sich somit aus der Anzahl der Spalten multipliziert mit der Anzahl der Window-Zeilen: 14*8 = 112

Der an der Window-Position liegende Text muß noch vor der Ausgabe des Windows gerettet werden, da er ja durch das Window überschrieben wird. Dies erreichen wir in den Zeilen 420–460. Wie Sie sehen, wird auch hier von 0 an numeriert, in den REM-Kommentaren stehen die entsprechenden Werte in der anderen Zählweise, die mit 1 beginnt. In Zeile 450 wird der jeweilige Wert aus dem Bildschirmspeicher ins Array T() eingelesen und der Index innerhalb des Arrays, die Variable I, erhöht.

Die Window-Ausgabe (Zeilen 480–580) bedarf keiner weiteren Erläuterung. In den Zeilen 650–690 wird dann das Array T() wieder in den Bildschirmspeicher geschrieben. An der Schleife ändert sich gegenüber 420–460 nur, daß jetzt der Wert nicht in T() geholt, sondern aus T() in den Bildschirmspeicher ge-POKE-t wird.

Nun ist diese Kopierschleife selbst bei recht kleinen Windows schon sehr langsam. Deshalb finden Sie als Programm »WINDOW-BSP. 3CMP« eine kompilierte Fassung, welche die volle Leistungsfähigkeit der Window-Technik nicht nur andeutet. Doch die endgültige Lösung heißt wieder einmal »Maschinensprache«!

8.2 Windows für Insider

Unser Ziel ist nun, Maschinenroutinen für Windows zu schreiben, die wir in ein Demoprogramm einbinden wollen. Wir werden zwei Routinen entwickeln; die eine nennen wir OUTW für »OUTput into Window« (Ausgabe eines Zeichens in ein Window), die andere PRWIN für »PRint WINdow« (Ausgabe eines kompletten Windows).
OUTW soll ähnlich wie die Betriebssystemroutine BSOUT ($FFD2) funktionieren (siehe »C 64 für Insider«), nur daß das auszugebende Zeichen in ein Window und nicht in den Normalbildschirm geschrieben wird; PRWIN soll ein ganzes Window in einem Zug drucken, so wie etwa beim C 128 die Routine PRIMM ($FF7D) einen kompletten Text ausgibt.
Für jedes der beiden angestrebten Windows schreiben wir eine eigene PRWIN-Routine (OUTW1/PRWIN1 für Window 1, OUTW2/PRWIN2 für Window 2). Diese Routinen halten wir so flexibel, daß sie leicht an andere Windows anzupassen sind.
Hier nun erhalten Sie die fertige Lösung als Quelltext vom Assembler Hypra-Ass, der sich ebenfalls auf der Rückseite der Programmdiskette befindet:

Listing 8.4: WINDOW-BSP. 4SRC

```
100  -.BA $C000
110  -;
120  -; ******************************
130  -; *                            *
140  -; *  MASCHINENSPRACHE-PROGRAMM *
150  -; *                            *
160  -; *      MIT WINDOW-TECHNIK    *
170  -; *                            *
180  -; *            C 64            *
190  -; *                            *
200  -; ******************************
210  -; *                            *
220  -; * (C) 1986 BY FLORIAN MUELLER *
230  -; *                            *
240  -; ******************************
250  -;
260  -          JMP MAIN       ; UNTERROUTINEN UEBERSPRINGEN
270  -;
280  -; SYMBOLDEFINITIONEN
290  -;
300  -.GL ZEILE   = $D6 ; AKTUELLE CURSORZEILE
310  -.GL SPALTE  = $D3 ; AKTUELLE CURSORSPALTE
320  -ZEILE1     .BY 0          ; ZEILE IN WINDOW #1
330  -SPALTE1    .BY 0          ; SPALTE I.WINDOW #1
340  -ZEILE2     .BY 0          ; ZEILE IN WINDOW #2
350  -SPALTE2    .BY 0          ; SPALTE I.WINDOW #2
360  -;
370  -.GL SETCUR  = $E50C        ; CURSOR SETZEN
380  -;
390  -.GL BASOUT = $FFD2         ; ZEICHEN AUSGEBEN
400  -.GL GET     = $FFE4        ; ZEICHEN EINGEBEN
410  -;
420  -TEMP1      .WO 0          ; 16-BIT-ZWISCHENSPEICHER
430  -ERKLFLAG   .BY 0          ; FLAG FUER ERKLAERUNG
440  -;
450  -.GL CLEAR = $93           ; ASCII-CODE FUER CLEAR
460  -.GL HOME  = $13           ; ASCII-CODE FUER HOME
```

```
470  -.GL CR      = $0D          ; ASCII-CODE FUER RETURN
480  -;
490  -.GL LINKS1  = 2            ; PARAMETER
500  -.GL RECHTS1 = 9+1          ; VON
510  -.GL OBEN1   = 2            ; WINDOW
520  -.GL UNTEN1  = 10+1         ; NR. 1
530  -;
540  -.GL LINKS2  = 12           ; PARAMETER
550  -.GL RECHTS2 = 29+1         ; VON
560  -.GL OBEN2   = 6            ; WINDOW
570  -.GL UNTEN2  = 18+1         ; NR. 2
580  -;
590  -.GL LAENGE1 = (RECHTS1-LINKS1)*(UNTEN1-OBEN1)
600  -.GL LAENGE2 = (RECHTS2-LINKS2)*(UNTEN2-OBEN2)
610  -;
620  -OUTPUTBYTE .BY 0           ; ZWISCHENSPEICHER
630  -;
640  -.GL TEXTPTR = $22          ; ZEIGER AUF TEXTSTELLE
650  -;
660  -; PROGRAMMTEXT:
670  -; =============
680  -;
690  -;
700  -;
710  -; UNTERROUTINEN FUER WINDOWS:
720  -; ===========================
730  -;
740  -OUTW1      STA OUTPUTBYTE ; ZEICHEN MERKEN
750  -           TXA            ; X-REGISTER
760  -           PHA            ; RETTEN
770  -           TYA            ; Y-REGISTER
780  -           PHA            ; RETTEN
790  -;
800  -           LDA SPALTE     ; CURSORSPALTE
810  -           PHA            ; RETTEN
820  -           LDA ZEILE      ; CURSORZEILE
830  -           PHA            ; RETTEN
840  -;
850  -           LDA OUTPUTBYTE ; ZEICHEN WIEDER HOLEN
860  -           CMP #CR        ; RETURN?
870  -           BNE TESTHOME1  ; NEIN, DANN WEITER
880  -           LDA ZEILE1     ; ZEILE HOLEN
890  -           CMP #UNTEN1    ; UNTERSTE ZEILE?
900  -           BCS ENDCR1     ; JA, DANN BEENDEN
910  -           INC ZEILE1     ; SONST ZEILE NACH UNTEN
920  -           LDA #LINKS1    ; LINKEN RAND ALS
930  -           STA SPALTE1    ; SPALTE SETZEN
940  -;
950  -ENDCR1     JMP ENDOUTW    ; WERTE VOM STACK HOLEN & ENDE
960  -;
970  -TESTHOME1  CMP #HOME      ; HOME?
980  -           BNE TESTCLR1   ; NEIN, DANN WEITER
990  -HOME1      LDA #OBEN1     ; OBEREN RAND
1000 -           STA ZEILE1     ; ALS ZEILE SETZEN
1010 -           LDA #LINKS1    ; LINKEN RAND
1020 -           STA SPALTE1    ; ALS SPALTE SETZEN
1030 -           JMP ENDOUTW    ; ENDE
1040 -;
1050 -TESTCLR1   CMP #CLEAR     ; CLEAR?
1060 -           BNE PRINT1     ; NEIN, DANN ZEICHEN AUSGEBEN
1070 -;
1080 -; WINDOW LOESCHEN:
1090 -;
1100 -           LDA #146       ; REVERS OFF
1110 -           JSR BASOUT     ; AUSGEBEN
1120 -;
1130 -           LDA #LINKS1    ; LINKE
```

```
1140 -           STA SPALTE1     ; OBERE
1150 -           LDA #OBEN1      ; ECKE ALS
1160 -           STA ZEILE1      ; CURSORPOSITION
1170 -;
1180 -SCHLEIFE1 JSR CLEAR1C      ; SPACE AUSGEBEN
1190 -           LDA SPALTE1     ;SPALTE HOLEN
1200 -           CMP #(RECHTS1-1); SCHON AM RECHTEN RAND?
1210 -           BCC CLEAR1A     ; NEIN, DANN SPALTE ERHOEHEN
1220 -           LDA ZEILE1      ; ZEILE HOLEN
1230 -           CMP #(UNTEN1-1); UNTERER RAND?
1240 -           BEQ ENDCLEAR1   ; JA, DANN BEENDEN
1250 -           BNE CLEAR1B     ; SONST AN BEGINN DER NAECHSTEN ZEILE
1260 -;
1270 -CLEAR1A    INC SPALTE1     ; SPALTE ERHOEHEN
1280 -           JMP SCHLEIFE1   ; WEITERMACHEN
1290 -;
1300 -CLEAR1B    INC ZEILE1      ; ZEILE ERHOEHEN
1310 -           LDA #LINKS1     ; LINKEN RAND
1320 -           STA SPALTE1     ; ALS SPALTE SETZEN
1330 -           JMP SCHLEIFE1   ; UND WEITER LOESCHEN
1340 -;
1350 -CLEAR1C    LDX ZEILE1      ; ZEILE
1360 -           LDY SPALTE1     ; UND SPALTE
1370 -           JSR SETCUR      ; SETZEN
1380 -           LDA #" "        ; LEERZEICHEN
1390 -           JMP BASOUT      ; AN LOESCHPOSITION AUSGEBEN
1400 -;
1410 -;
1420 -ENDCLEAR1 JMP HOME1        ; CURSOR IN LINKE OBERE WINDOW-ECKE
1430 -;
1440 -;
1450 -;
1460 -PRINT1     LDX ZEILE1      ; ZEILE
1470 -           LDY SPALTE1     ; UND SPALTE
1480 -           JSR SETCUR      ; IM WINDOW SETZEN
1490 -           LDA OUTPUTBYTE  ; ZEICHEN HOLEN
1500 -           JSR BASOUT      ; UND AN GESETZTER POSITION AUSGEBEN
1510 -;
1520 -           INC SPALTE1     ; SPALTE ERHOEHEN
1530 -           LDA SPALTE1     ; SPALTE HOLEN
1540 -           CMP #RECHTS1    ; UEBER RECHTE GRENZE HINAUS?
1550 -           BCC ENDOUTW     ; NEIN, DANN BEENDEN
1560 -           LDA #LINKS1     ; SONST LINKEN RAND
1570 -           STA SPALTE1     ; ALS SPALTE SETZEN
1580 -           INC ZEILE1      ; EINE ZEILE NACH UNTEN
1590 -           LDA ZEILE1      ; ZEILE HOLEN
1600 -           CMP #UNTEN1     ; SCHON UNTERSTE ZEILE
1610 -           BCC ENDOUTW     ; NEIN, DANN ENDE
1620 -           LDA #OBEN1      ; SONST CURSOR IM WINDOW
1630 -           STA ZEILE1      ; IN OBERSTE ZEILE
1640 -;
1650 -ENDOUTW    PLA             ; ZEILE
1660 -           STA ZEILE       ; HOLEN
1670 -           PLA             ; SPALTE
1680 -           STA SPALTE      ; HOLEN
1690 -;
1700 -           PLA             ; Y-REGISTER
1710 -           TAY             ; HOLEN
1720 -           PLA             ; X-REGISTER
1730 -           TAX             ; HOLEN
1740 -           LDA OUTPUTBYTE  ; ZEICHEN IN AKKU
1750 -           RTS             ; ROUTINE BEENDEN
1760 -;
1770 -;
1780 -; DIE GLEICHE ROUTINE FUER
1790 -; WINDOW #2:
1800 -;
```

```
1810 -OUTW2      STA OUTPUTBYTE
1820 -           TXA
1830 -           PHA
1840 -           TYA
1850 -           PHA
1860 -;
1870 -           LDA SPALTE
1880 -           PHA
1890 -           LDA ZEILE
1900 -           PHA
1910 -;
1920 -           LDA OUTPUTBYTE
1930 -           CMP #CR
1940 -           BNE TESTHOME2
1950 -           LDA ZEILE2
1960 -           CMP #UNTEN2
1970 -           BCS ENDCR2
1980 -           INC ZEILE2
1990 -           LDA #LINKS2
2000 -           STA SPALTE2
2010 -;
2020 -ENDCR2     JMP ENDOUTW
2030 -;
2040 -TESTHOME2 CMP #HOME
2050 -           BNE TESTCLR2
2060 -HOME2      LDA #OBEN2
2070 -           STA ZEILE2
2080 -           LDA #LINKS2
2090 -           STA SPALTE2
2100 -           JMP ENDOUTW
2110 -;
2120 -TESTCLR2   CMP #CLEAR
2130 -           BNE PRINT2
2140 -;
2150 -; WINDOW LOESCHEN:
2160 -;
2170 -           LDA #146        ; REVERS OFF
2180 -           JSR BASOUT      ; AUSGEBEN
2190 -;
2200 -           LDA #LINKS2
2210 -           STA SPALTE2
2220 -           LDA #OBEN2
2230 -           STA ZEILE2
2240 -;
2250 -SCHLEIFE2 JSR CLEAR2C
2260 -           LDA SPALTE2
2270 -           CMP #(RECHTS2-1)
2280 -           BCC CLEAR2A
2290 -           LDA ZEILE2
2300 -           CMP #(UNTEN2-1)
2310 -           BEQ ENDCLEAR2
2320 -           BNE CLEAR2B
2330 -;
2340 -CLEAR2A    INC SPALTE2
2350 -           JMP SCHLEIFE2
2360 -;
2370 -CLEAR2B    INC ZEILE2
2380 -           LDA #LINKS2
2390 -           STA SPALTE2
2400 -           JMP SCHLEIFE2
2410 -;
2420 -CLEAR2C    LDX ZEILE2
2430 -           LDY SPALTE2
2440 -           JSR SETCUR
2450 -           LDA #" "
2460 -           JMP BASOUT
2470 -;
```

```
2480 -;
2490 -ENDCLEAR2 JMP HOME2
2500 -;
2510 -;
2520 -;
2530 -PRINT2    LDX ZEILE2
2540 -          LDY SPALTE2
2550 -          JSR SETCUR
2560 -          LDA OUTPUTBYTE
2570 -          JSR BASOUT
2580 -;
2590 -          INC SPALTE2
2600 -          LDA SPALTE2
2610 -          CMP #RECHTS2
2620 -          BCC ENDOUTW2
2630 -          LDA #LINKS2
2640 -          STA SPALTE2
2650 -          INC ZEILE2
2660 -          LDA ZEILE2
2670 -          CMP #UNTEN2
2680 -          BCC ENDOUTW2
2690 -          LDA #OBEN2
2700 -          STA ZEILE2
2710 -;
2720 -ENDOUTW2  JMP ENDOUTW     ; WEITER WIE BEI OUTW1
2730 -;
2740 -;
2750 -;
2760 -;
2770 -; ROUTINE ZUM DRUCKEN EINES
2780 -; GANZEN WINDOWS
2790 -;
2800 -; ADRESSE WIRD IN A/Y UEBERGEBEN
2810 -;
2820 -PRWIN1    STA TEXTPTR     ; LO-BYTE SETZEN
2830 -          STY TEXTPTR+1   ; HI-BYTE SETZEN
2840 -;
2850 -          LDA #HOME       ; LINKE OBERE WINDOW-ECKE
2860 -          JSR OUTW1       ; ANSPRINGEN
2870 -          CLC             ; CARRY VOR ADDITION LOESCHEN
2880 -          LDA TEXTPTR     ; ZUR ANFANGS-
2890 -          ADC #<(LAENGE1); ADRESSE DES
2900 -          STA TEMP1       ; TEXTES
2910 -          LDA TEXTPTR+1   ; WIRD LAENGE DES
2920 -          ADC #>(LAENGE1); TEXTES ADDIERT.
2930 -          STA TEMP1+1     ; ERGEBNIS IN (TEMP1)/(TEMP1+1)
2940 -;
2950 -          LDY #0          ; OFFSET = 0
2960 -SCHLEIFE3 LDA (TEXTPTR),Y; ZEICHEN AUS TEXT HOLEN
2970 -          JSR OUTW1       ; UND AUSGEBEN
2980 -          INC TEXTPTR     ; ZEIGER AUF
2990 -          BNE PRWIN1A     ; TEXTSTELLE
3000 -          INC TEXTPTR+1   ; UM 1 ERHOEHEN
3010 -;
3020 -PRWIN1A   LDA TEXTPTR     ; PRUEFEN,
3030 -          CMP TEMP1       ; OB SCHON
3040 -          LDA TEXTPTR+1   ; DER GANZE
3050 -          SBC TEMP1+1     ; TEXT AUSGEGEBEN WURDE
3060 -          BCC SCHLEIFE3   ; NEIN (C=0): WEITER
3070 -;
3080 -          RTS             ; SONST ENDE
3090 -;
3100 -;
3110 -; ERKLAERUNGSTEXT IN WINDOW #1
3120 -; SCHREIBEN
3130 -;
3140 -;
```

```
3150 -ERKLAERG  LDA #18         ; REVERS ON
3160 -          JSR BASOUT      ; AUSGEBEN
3170 -          LDA #<(ERKLTX)  ; ERKLAERUNGS-
3180 -          LDY #>(ERKLTX)  ; TEXT
3190 -          JMP PRWIN1      ; AUSGEBEN
3200 -;
3210 -ERKLTX    .TX "         "
3220 -          .TX " DIES    "
3230 -          .TX "         "
3240 -          .TX " IST     "
3250 -          .TX "         "
3260 -          .TX " WINDOW  "
3270 -          .TX "         "
3280 -          .TX " NR. 1   "
3290 -          .TX "         "
3300 -;
3310 -;
3320 -; ROUTINE ZUM DRUCKEN EINES
3330 -; GANZEN WINDOWS FUER WINDOW #2
3340 -;
3350 -PRWIN2    STA TEXTPTR
3360 -          STY TEXTPTR+1
3370 -;
3380 -          LDA #HOME
3390 -          JSR OUTW2
3400 -          CLC
3410 -          LDA TEXTPTR
3420 -          ADC #<(LAENGE2)
3430 -          STA TEMP1
3440 -          LDA TEXTPTR+1
3450 -          ADC #>(LAENGE2)
3460 -          STA TEMP1+1
3470 -;
3480 -          LDY #0
3490 -SCHLEIFE4 LDA (TEXTPTR),Y
3500 -          JSR OUTW2
3510 -          INC TEXTPTR
3520 -          BNE PRWIN2A
3530 -          INC TEXTPTR+1
3540 -;
3550 -PRWIN2A   LDA TEXTPTR
3560 -          CMP TEMP1
3570 -          LDA TEXTPTR+1
3580 -          SBC TEMP1+1
3590 -          BCC SCHLEIFE4
3600 -;
3610 -          RTS
3620 -;
3630 -;
3640 -; MENUE IN WINDOW #2 SCHREIBEN
3650 -;
3660 -;
3670 -SCHRMENUE LDA #146        ; REVERS OFF
3680 -          JSR BASOUT      ; AUSGEBEN
3690 -          LDA #<(MENUETX) ; MENUE-
3700 -          LDY #>(MENUETX) ; TEXT
3710 -          JMP PRWIN2      ; AUSGEBEN
3720 -;
3730 -MENUETX   .TX "******************"
3740 -          .TX "*                *"
3750 -          .TX "*     MENUE      *"
3760 -          .TX "*                *"
3770 -          .TX "******************"
3780 -          .TX "*                *"
3790 -          .TX "* (1) ERKLAERUNG *"
3800 -          .TX "*                *"
3810 -          .TX "* (2) FARBE      *"
```

```
3820 -            .TX "*                  *"
3830 -            .TX "*  (3) BEENDEN     *"
3840 -            .TX "*                  *"
3850 -            .TX "********************"
3860 -;
3870 -;
3880 -;
3890 -; HAUPTPROGRAMM
3900 -; -------------
3910 -;
3920 -MAIN        LDA #0          ; CODE FUER SCHWARZ
3930 -            STA $D020       ; RAHMEN SCHWARZ
3940 -            STA $D021       ; HINTERGRUND SCHWARZ
3950 -            LDA #5          ; ASCII-CODE FUER WEISS
3960 -            JSR BASOUT      ; ALS SCHRIFTFARBE SETZEN
3970 -            LDA #0          ; ERKLAERUNGSFLAG
3980 -            STA ERKLFLAG    ; ZURUECKSETZEN
3990 -;
4000 -            JSR SCHRMENUE   ; MENUE AUSGEBEN
4010 -;
4020 -TASTE       JSR GET         ; TASTENDRUCK HOLEN
4030 -            CMP #"1"        ; "1"?
4040 -            BEQ PUNKT1      ; JA: ROUTINE FUER PUNKT 1 ANSPRINGEN
4050 -            CMP #"2"        ; "2"?
4060 -            BEQ PUNKT2      ; JA: ROUTINE FUER PUNKT 2 ANSPRINGEN
4070 -            CMP #"3"        ; "3"?
4080 -            BNE TASTE       ; NEIN, DANN AUF TASTE WARTEN
4090 -;
4100 -; PUNKT 3 (BEENDEN):
4110 -;
4120 -            RTS             ; BEENDEN
4130 -;
4140 -;
4150 -; PUNKT 2 (FARBE):
4160 -;
4170 -PUNKT2      INC $D020       ; RAHMENFARBE ERHOEHEN
4180 -            INC $D021       ; AUCH HINTERGRUNDFARBE
4190 -            JMP TASTE       ; ZURUECK ZUM MENUE
4200 -;
4210 -;
4220 -PUNKT1      LDA ERKLFLAG    ; ERKLAERUNGSFLAG
4230 -            EOR #1          ; "SKIPPEN"
4240 -            STA ERKLFLAG    ; UND ZURUECKSCHREIBEN
4250 -;
4260 -            BEQ LOESCHERK   ; =0, DANN WINDOW #1 LOESCHEN
4270 -;
4280 -            JSR ERKLAERG    ; SONST ERKLAERUNG AUSGEBEN
4290 -            JMP TASTE       ; UND ZURUECK ZUM MENUE
4300 -;
4310 -LOESCHERK   LDA #CLEAR      ; WINDOW #1
4320 -            JSR OUTW1       ; LOESCHEN
4330 -            JMP TASTE       ; ZURUECK ZUM MENUE
```

Der lauffähige Objektcode wird über LOAD "WINDOW-BSP. 4OBJ",8,1 geladen und nach NEW mit SYS 49152 gestartet. Daraufhin erscheint in Window #1 das Menü. Mit <1> wird eine Erklärung ein-/ausgeblendet, mit <2> die Bildschirmfarbe geändert und mit <3> das Programm beendet. Die Erklärung erscheint ebenfalls, wie sollte es auch anders sein, in einem Window, nämlich Window #2.

Nun zum Quelltext. Zunächst werden darin die Symbole definiert. ZEILE und SPALTE sind Zeropage-Adressen, in denen die aktuelle Cursorzeile und -spalte stehen (siehe »C 64 für Insider«); da es sich um Betriebssystemvariablen handelt, läßt sich daran zu jedem Zeitpunkt die Cursorposition feststellen.

ZEILE1/SPALTE1 beziehungsweise ZEILE2/SPALTE2 sind die Cursorpositionen innerhalb der Windows. Diese werden vom Programm laufend neu berechnet. SETCUR ist ein Betriebssystem-Einsprung zur Cursorpositionierung, welcher in »C 64 für Insider« als Bestandteil von PLOT ($FFF0) aufgeführt ist; bei SETCUR wird im X-Register die Zeile, im Y-Register die Spalte übergeben.

BASOUT und GET sind Kernel-Routinen zur Aus- beziehungsweise Eingabe eines Zeichens. TEMP1 ist ein 2-Byte-Zwischenspeicher, in welchem die PRWIN1/PRWIN2-Routine die Endadresse des auszugebenden Textes ablegt. ERKLFLAG ist ein Flag dafür, ob der Erklärungstext am Bildschirm steht (Inhalt von ERKLFLAG = 0) oder ob er nicht angezeigt ist (Inhalt von ERKLFLAG = 1). CLEAR, HOME und CR sind die ASCII-Codes der gleichnamigen Steuerzeichen.

LINKS1/ RECHTS1/ OBEN1/ UNTEN1 bzw. LINKS2/ RECHTS2/ OBEN2/ UNTEN2 sind die Grenzen der Windows. LINKS1/LINKS2 und OBEN1/OBEN2 werden in der mit 0 beginnenden Numerierung angegeben. Zu RECHTS1/RECHTS2 und UNTEN1/UNTEN2 addieren wir noch 1 (siehe »+1« in 500, 520, 550, 570).

Aus diesen Parametern können wir dann in 590/600 die Anzahl der Zeichen in einem Window berechnen und in LAENGE1 bzw. LAENGE2 vermerken. Diese Länge ist für die PRWIN-Routinen wichtig, damit sofort feststeht, wie viele Zeichen ausgegeben werden müssen. OUTPUTBYTE (Zeile 620) ist ein Zwischenspeicher, in dem das über eine OUTW-Routine auszugebende Zeichen gespeichert wird. TEXTPTR ist ein Zeropage-Zeiger für unsere Zwecke, den wir von der PRWIN-Routine auf das auszugebende Zeichen stellen lassen.

Mit diesen Erläuterungen und den Kommentaren im Quelltext dürfte es für Sie kein Problem sein, das Programm zu verstehen; eine Systemdokumentation wäre sicherlich hilfreich (siehe »C 64 für Insider«). Lediglich zu Zeile 4230 ist noch zu sagen, daß hier das ERKLFLAG von 0 auf 1 bzw. von 1 auf 0 umgeschaltet wird.

Damit Sie die Routinen auch für eigene Programme nutzen können, wollen wir die Parameterübergabe an OUTW/PRWIN besprechen, wie in einem Assembler-Kurs die Benutzung einer Betriebssystem-Routine besprochen wird. Ich halte mich dabei weitestgehend an das Schema aus meinem Buch »C 64 für Insider«:

```
OUTW (OUTW1 ab Zeile 740, OUTW2 ab Zeile 1810):
     Ausgabe eines Zeichens in ein Window
```

Diese Routine gibt ein Zeichen, dessen ASCII-Code im Akkumulator steht, in das betreffende Window aus:

```
LDA #character
JSR OUTW;        OUTW1 oder OUTW2
```

Nach Aufruf der Routine besitzen Akku, X- und Y-Register wieder denselben Wert wie vor dem JSR-OUTW-Befehl.
Für den ASCII-Code im Akkumulator sind alle druckenden Zeichen, d.h. alle Zeichen außer den Steuerzeichen, vorgesehen. Die meisten Steuerzeichen führen zu Fehlfunktionen; dennoch sind die Steuercodes CLEAR ($93), HOME ($13) und CR ($0D) zugelassen und wurden für die Window-Technik modifiziert:

- HOME springt die linke obere Ecke des Windows an
- CLEAR löscht das Window und bewegt den Cursor zu dessen linker oberer Ecke
- CR springt an den Anfang der nächsten Zeile im Window

Zur Ausgabe von Farbsteuerzeichen oder Anweisungen wie RVS ON und RVS OFF ist die Kernel-Routine BSOUT ($FFD2) aufzurufen; die dadurch hervorgerufenen Effekte werden von OUTW beibehalten, ein über BSOUT ($FFD2) bewirkter Reversdruck gilt also auch für die von OUTW ausgegebenen Zeichen. Letztlich stützt sich OUTW nämlich auf BSOUT ($FFD2).

Hinweis: Bei erstem Aufruf von OUTW sollte man den Cursor in die HOME-Position bringen oder das Window mit CLEAR löschen. Die Cursorposition steht jeweils in ZEILE1/SPALTE1 bzw. ZEILE2/SPALTE2, wobei der dort stehende Ausgangswert 0/0 nur in seltenen Fällen sinnvoll ist.

```
PRWIN (PRWIN1 ab Zeile 2820, PRWIN2 ab Zeile 3350):
Ausgabe eines kompletten Windows
```

Um ganze Window-Inhalte auf einen Schlag zu drucken, lädt man die Adresse des Textes in Akkumulator (Low-Byte) und Y-Register (High-Byte). Ab dieser Adresse hat lediglich der Window-Inhalt als Folge von ASCII-Codes zu stehen; am Ende des Textes und mitten im Text sind keinerlei Endmarkierungen erforderlich, denn aus den Window-Grenzen geht eindeutig hervor, wie viele Zeichen ausgegeben werden müssen. Hier werden die Label LAENGE1 und LAENGE2 herangezogen.
Zur Funktionsweise: PRWIN stützt sich auf OUTW.
Zwei Anwendungen von PRWIN sehen Sie in den Zeilen 3150–3300 und 3670–3860.

```
MAIN (ab Zeile 3920):
Hauptprogramm des Demos
```

Das Hauptprogramm MAIN ist nur eine Demonstration der Window-Technik. Das Grundlegende sind jedoch die Routinen zur Window-Technik. Wenn Sie diese in Ihren Quelltext übernehmen wollen, müssen Sie auch die darin verwendeten Label übernehmen; die Berechnungsformel für LAENGE (Zeilen 590/600) sollten Sie ebenfalls verwenden, allerdings ist hier kein Umschreiben nötig.
Das Hauptprogramm MAIN hingegen können Sie nach Belieben durch eigene Rahmenprogramme ersetzen.

9 Bildschirmgestaltung mit Maskengenerator oder: Laß andere arbeiten

Die ersten Schritte eines Basic-Anfängers bestehen zweifelsohne in der »Programmierung« von PRINT-Zeilen. Auch mir ist es im Anfangsstadium meiner Computerbegeisterung so gegangen, daß ich ge-PRINT-et habe, bis mir die Augen tränten. Doch bei fortgeschrittenen Programmierkenntnissen schlägt man sich oft damit herum, eine ansprechende Bildschirmausgabe zu entwickeln. Hier erhalten Sie ein wirklich fantastisches Utility, das bei der täglichen Programmierung unübersehbare Vorteile bietet.

9.1 Das Prinzip von Maskengeneratoren

Für die unkomplizierte und komfortable Erzeugung von PRINT-Zeilen gibt es sogenannte Maskengeneratoren, also Programme, die eine »Maske« (Bildschirminhalt) einlesen und in fertige PRINT-Befehle umwandeln. Diese Idee ist zwar im Ansatz sehr gut, hatte aber bislang in der Praxis einen erheblichen Nachteil aufzuweisen: Es wurden nur PRINT-Anweisungen erzeugt, nicht aber Befehle wie FOR, TO, NEXT, CHR$, SPC und POKE, deren Anwendung besonders dann notwendig wird, wenn der Speicherplatz nur begrenzt ist. Deshalb erhalten Sie hier ein Programm, welches über die nötige »Intelligenz« verfügt, effektive Ausgabeprogramme zu generieren.

Vom nun vorgestellten Maskengenerator werden verschiedene Schriftfarben ebenso berücksichtigt wie die Rahmen- und Hintergrundfarbe, invers dargestellte Zeichen und sogar die Wahl des Zeichensatzes (Groß- oder Kleinschrift). Im Programm ist außerdem ein Editor integriert, der eine schnellere und bequemere Eingabe von Masken ermöglicht, als es mit dem Basic-Editor jemals möglich wäre.

Die Arbeitsgeschwindigkeit ist enorm: Es dauert nur etwa 1 Sekunde, bis eine Bildschirmmaske in Form von Befehlszeilen im Speicher steht. Und was die allgemeine Arbeitserleichterung betrifft, so arbeitet man mit dem Maskengenerator so unglaublich effektiv, daß man bei der Entwicklung von Programmen mit vielen Bildschirmausgaben sogar zeitliche Einsparungen im Stundenbereich erzielt.

9.2 Die Bedienung des Maskengenerators

Um bei der Programmierung auf den Maskengenerator von Georg Wichert zurückgreifen zu können, laden Sie ihn von der Rückseite der Masterdiskette über LOAD"MASKENGEN.49152",8,1 und geben zunächst NEW ein, um die Basic-Programmeingabe wieder zu ermöglichen. Bei Bedarf starten Sie den Maskengenerator über folgendes Kommando:

```
SYS 49152,startzeile,schrittweite
```

Für »startzeile« ist dabei die Zeilennummer einzusetzen, ab welcher die generierten PRINT-Zeilen stehen sollen; die Numerierung der Zeilen ist abhängig von »schrittweite«, wofür nur Werte von 0 bis 256 zulässig und sinnvoll sind.
Nach dem Start über den genannten Befehl geben Sie nun Ihre Maske ein, wofür Ihnen zahlreiche Steuerbefehle (siehe 9.3) zur Verfügung stehen. Neben den Steuerbefehlen kann man auf gewohnte Weise jede der 16 möglichen Schriftfarben anwählen; alle Zeichen und Buchstaben sind uneingeschränkt zu verwenden, was auch für Anführungszeichen gilt. Ist die Eingabe beendet, so geben Sie <RUN/STOP> und anschließend <Y> ein; innerhalb einer Sekunde wird nun das Maskenprogramm erzeugt. Dieses kann auch nachträglich im Basic-Betrieb modifiziert werden.

9.3 Die Steuerbefehle des Maskengenerators

Folgende Tasten sind neben den Farbansteuerungen und den druckenden Zeichen zulässig:

```
<CTRL C> Centre/Zentrierung
```

Der Inhalt der Cursorzeile wird in die Mitte verschoben.

```
<CTRL D> Delete Line/Zeile löschen
```

Die aktuelle Cursorzeile und alle Zeilen darunter werden um eine Zeile nach oben verschoben, die unterste Zeile geht dabei verloren. Letztlich wird die Zeile über der aktuellen Cursorzeile frei für neue Eingaben.

```
<CTRL I> Insert Line/Zeile einfügen
```

Die aktuelle Cursorzeile und alle Zeilen darunter werden um eine Zeile nach unten verschoben, die aktuelle Cursorzeile wird also letztlich frei für neue Eingaben.

`<RUN/STOP> Finish/Beenden`

Beendet die Eingabe. Davor erfolgt die Sicherheitsabfrage »finished?«, welche mit <Y> (= YES) oder <N> (= NO) zu beantworten ist. Man sollte nicht zu voreilig <Y> auslösen, denn nach einmaligem Generieren eines Maskenprogramms läßt sich dieses nur noch im Basic-Betrieb editieren – und das ist erheblich unkomfortabler und schwieriger als eine Bearbeitung mit dem Maskengenerator.

`<RETURN> Carriage Return/Zeilenrücksprung`

Der Cursor springt zum Anfang der nächsten Zeile.

`<CRSR DOWN> Cursor Down/Cursor nach unten`

Der Cursor geht um eine Zeile nach unten; aus der untersten Zeile springt er in die oberste zurück. Das Bildschirm-Scrolling, welches die bestehenden Bildschirminhalte in verheerender Weise löschen kann, findet also nicht statt.

`<CRSR UP> Cursor Up/Cursor nach oben`

Der Cursor geht um eine Zeile nach oben. In der obersten Zeile erfolgt ein Sprung in die unterste.

`<CRSR RIGHT> Cursor Right/Cursor nach rechts`

Der Cursor bewegt sich um eine Position nach rechts.

`<CRSR LEFT> Cursor Left/Cursor nach links`

Der Cursor bewegt sich um eine Position nach links.

`<HOME> Cursor Home/Cursor in linke obere Ecke`

Der Cursor springt in die HOME-Position, die linke obere Ecke des Bildschirms.

`<DEL> Delete/Löschen`

Das links vom Cursor stehende Zeichen wird gelöscht und der Rest der Zeile um eine Position nach links verschoben.

`<INST> Insert/Einfügen`

Die Cursorposition und der Rest der Zeile werden innerhalb einer Zeile um eine Position nach rechts geschoben, wobei an die Einfügestelle ein Leerzeichen kommt.

<CTRL X> Delete in X direction/Löschen in X-Richtung

Die aktuelle Cursorzeile (40 Zeichen) wird gelöscht.

<CTRL Y> Delete in Y direction/Löschen in Y-Richtung

Die aktuelle Cursorspalte (25 Zeichen) wird gelöscht.

<F1> Border/Rahmen

Die Rahmenfarbe wird verändert.

<F3> Background/Hintergrund

Die Hintergrundfarbe wird verändert.

<F5> Repeat all keys/Dauerfunktion

Dies schaltet die Wiederholungsfunktion (Dauerfunktion, Repeat), die sich auf alle (!) Tasten bezieht, ein und aus.

<F7> RVS switch/Revers-Schaltung

Damit wird die inverse Darstellung der einzugebenden Zeichen ein- und wieder ausgeschaltet.

<CLR> Clear Screen/Bildschirm löschen

Damit Sie beim Löschen des Bildschirms Ihre wertvolle Arbeit nicht versehentlich vernichten, erfolgt eine Sicherheitsabfrage »clear screen?«, welche mit <Y> (= YES) oder <N> (= NO) zu beantworten ist.

<C= SHIFT> Switch Font/Zeichensatzumschaltung

Damit wird zwischen Groß- und Kleinschrift umgeschaltet.

10 Effektives Programmieren in Assembler oder: Power für Profis

Es gibt viele Möglichkeiten, ein Basic-Programm schneller und komfortabler zu gestalten; weite Teile des Buches widmen sich dieser Aufgabe. Aber auch für Maschinenprogrammierung gibt es einige Tricks und Kniffe, die ich Ihnen in diesem ausführlichen Kapitel verrate. Dabei möchte ich auch alle diejenigen einladen, die noch nicht so sehr mit Maschinensprache vertraut sind; denn gerade die hier vorgestellten Tricks stellen oft einfache Fertiglösungen dar, die speziell für den Einstieg in Assembler wertvoll sind.
Die meisten dieser Tricks habe ich nämlich in meinen C64-Anfangszeiten vor längerer Zeit entwickelt; dennoch werden Sie zu den angesprochenen Themen in keiner anderen Literatur vergleichbare Informationen finden!
Sie erfahren hier, wie man

a. Programme beschleunigen
b. komfortable Menüs schaffen
c. Speicherplatz sparen kann

Die Beispielprogramme sind anhand des Filenamens zu erkennen:

(B)	=	Basic-Programm
(S)	=	Quelltext vom Assembler Hypra-Ass, welcher sich ebenfalls auf der Diskette befindet (Beschreibung im 64'er-Sonderheft 8/85)
(O)	=	ablauffähiger Objektcode; dem Quelltext ist die Startadresse zu entnehmen
(L)	=	nur als Listing, nicht als Hypra-Ass-Quelltext
kein Zusatz	=	Objektcode, zu welchem kein Quellcode existiert

Die meisten Programme erhalten Sie in doppelter Ausführung: Quelltext (S) zur Dokumentation bzw. Modifikation und Objektcode (O) zum Testen bzw. Kennenlernen.

10.1 Systembeschleunigungen

Der C64 muß viele Aufgaben »gleichzeitig« erledigen: Bearbeitung des Hauptprogramms, Ablauf des Systeminterrupts und Senden des Video-Signals (an den Monitor/Fernseher). Alle diese Funktionen erfordern

- viele Zugriffe auf den Datenbus des Prozessors
- und (teilweise dadurch bedingt) Ausführungszeit

Unser Grundproblem ist nun in diesem Abschnitt, wie wir den C64 dazu bewegen, diese Aufgaben nicht (oder nur teilweise) auszuführen.

10.1.1 Beschleunigungen des Systems in Assembler

a. Eingriffe in den Systeminterrupt

Eine detaillierte Beschreibung des Systeminterrupts finden Sie im oft erwähnten Buch »C64 für Insider«. Hier möchte ich nur kurz zusammenfassen, was im normalen Interrupt des Betriebssystems geschieht: 60mal in der Sekunde wird das Hauptprogramm verlassen und die Routine ab $EA31 angesprungen. Ist diese abgearbeitet, wird wieder ins Hauptprogramm zurückgekehrt. Während dieser Unterbrechung (»interrupt«) tut sich einiges:

- die RUN/STOP-Taste wird überprüft
- die Tastatur und der Datasettenmotor werden abgefragt
- das Cursorblinken wird erledigt
- die interne Uhr (TI$) wird gestellt

Überlegen wir uns, welche Funktionen verzichtbar sind:

- Die RUN/STOP-Taste bewirkt nur in Basic-Programmen einen Abbruch, in Assembler müßte sie zum Beispiel über »JSR $FFE1« zusätzlich abgefragt werden.
- Die interne Uhr findet von Maschinensprache aus nur selten Verwendung, da die CIA-Timer hierfür geeigneter sind.

Kurz und gut, ein Maschinenprogramm kann durchaus auf beide Funktionen verzichten. Dies wird durch

```
LDA #$34
STA $0314
```

erreicht. Weil der Computer dadurch von Nebenaufgaben entlastet wird, läuft das Hauptprogramm etwas schneller ab. Die Normaleinstellung erhält man wieder mit

```
LDA #$31
STA $0314
```

Beschleunigungsmethode 1

Trick: Verkürzung der Interrupt-Routine
Nebenwirkungen: Abfrage der STOP-Taste und interne Uhr entfallen

Können Sie zwischenzeitlich auf die ganze Interrupt-Routine verzichten, genügt ein einziger Befehl: SEI verhindert grundsätzlich das Auftreten von Interrupts; CLI bewirkt die Normaleinstellung.

Beschleunigungsmethode 2

Trick: Interrupt total abschalten
Nebenwirkungen: Abfrage von Tastatur, STOP-Taste und Datasettenmotor sowie Cursor und interne Uhr entfallen.

Es gibt aber noch eine Möglichkeit im Zusammenhang mit dem Systeminterrupt: Von der Adresse $DC05, die als Zähler dient, hängt die Anzahl der Interrupts (in der Regel: 60 Aufrufe pro Sekunde) innerhalb eines bestimmten Zeitintervalls ab. Diese Adresse läßt sich durch Schreibzugriff ändern. Schreibt man in $DC05 einen niedrigen Wert (im Extremfall 0), so werden sehr viele Interrupts ausgelöst. Dies macht sich in der Geschwindigkeit der IRQ-Routine bemerkbar: Cursor und Tastaturabfrage werden hyperschnell, die interne Uhr geht deutlich vor, und so weiter. Verwendet man eine eigene, eventuell zeitkritische Interrupt-Routine, so kann sie auf diese Weise beschleunigt werden.
Dieser Geschwindigkeitszuwachs geht allerdings auf Kosten des Hauptprogramms, das stark verlangsamt wird. Bei wenigen Interrupts (große Zahl in $DC05) wird es beschleunigt. Die entsprechenden Assemblerbefehle lauten

```
LDA #$FF
STA $DC05
```

um eine starke Beschleunigung zu bewirken. Die Normaleinstellung wird durch

```
LDA #$3A
STA $DC05
```

erreicht.

Beschleunigungsmethode 3

Trick: Anzahl der Interruptaufrufe pro Sekunde ändern
Nebenwirkungen: Bei zu wenigen Aufrufen hinken Uhr, Cursor und Tastaturabfrage nach; bei zu vielen sind sie zu schnell.

---

b. VIC-Register #17

Ist Ihnen schon beim Arbeiten mit der Datasette oder einigen Kopierprogrammen aufgefallen, daß manchmal der Bildschirm abgeschaltet wird (ähnlich wie im FAST-Mode des C 128)? Dies kann man mit einem Vorhang vergleichen, der zwischenzeitlich den Bildchirm verdeckt. Der Bildschirm kann zwar nach wie vor (hinter dem Vorhang, versteht sich) geändert werden, d.h., Ausgabeoperationen werden ausgeführt, aber sichtbar wird die Wirkung erst, wenn der Vorhang entfernt wird.
Verantwortlich für das Ein-/Ausschalten des Bildschirms ist das VIC-Register #17:

Bit 4 gesetzt:	Bildschirm wird angezeigt
Bit 4 gelöscht:	Bildschirm wird abgeschaltet (und nimmt Rahmenfarbe an)

Da wir die theoretischen Grundlagen haben, brauchen wir unser Wissen nur noch in Befehle umzusetzen.

Bildschirm abschalten:

```
LDA $D011          ; $D011 ist VIC-Register #17
AND #$EF           ; $EF = %11101111
STA $D011          ; gelöschtes Bit 4 zurückschreiben
```

In diesem Zustand laufen Operationen zirka 5 Prozent schneller ab. Doch es sei nur am Rande erwähnt, daß dies bei Floppy-Betrieb nichts bringt; wenn Floppy-Zugriffe bei abgeschaltetem Bildschirm erfolgen, so geschieht dies aus anderen Gründen (Verhinderung von Timing-Problemen durch VIC-Zugriffe auf den Datenbus).

Bildschirm wieder einschalten:

```
LDA $D011
ORA #$10           ; $10 = %00010000
STA $D011          ; gesetztes Bit 4 zurückschreiben
```

Dies ist der Normalzustand.

Beschleunigungsmethode 4

Trick: Bildschirm abschalten
Nebenwirkungen: Der Bildschirminhalt ist nicht zu sehen, geht aber auch nicht verloren.

Im C64-Modus des C128 ist nun noch eine sensationelle Beschleunigung um einen Faktor von etwa 55 Prozent zu erzielen, wenn in Adresse #53296 ($D030) Bit 0 gesetzt wird:

```
LDA #$01
STA $D030
```

Dabei ist zusätzlich der Bildschirm abzuschalten, da er ansonsten ein fürchterliches Aussehen annimmt. Den Normalzustand erreicht man über

```
LDA #$00
STA $D030
```

Den Programmierern, die sich für derartige C128-Besonderheiten und eine endgültige Information über den C64-Modus des C128 interessieren, lege ich deshalb spezielle C128-Literatur ans Herz (»Vom C64 zum C128 – Tips & Tricks«, Kapitel 6).

Beschleunigungsmethode 5

Trick: FAST-Modus einschalten (nur im C64-Modus des C128 möglich)
Nebenwirkungen: Der Bildschirminhalt flackert erheblich, wenn man nicht Beschleunigungsmethode 4 anwendet. Zugriffe auf die Floppy dürfen nicht erfolgen.

c. Hinweise zum bisher Gesagten

Alle bis zu dieser Stelle genannten Tricks beziehen sich auf die Beschleunigung von Programmen. Sie lassen sich leicht nachträglich einfügen, weil am eigentlichen Programmalgorithmus keine Änderungen erforderlich sind. Sie können das Abschalten des Bildschirms mit dem Abschalten oder Einschränken des Interrupts verknüpfen, um die Geschwindigkeit noch weiter zu erhöhen. Wenn Sie hingegen den Interrupt ganz abschalten (SEI), bringt es naturgemäß keinen zusätzlichen Gewinn, ihn einzuschränken oder die Zahl der Aufrufe zu ändern. Beachten Sie bitte, daß alle beschriebenen Tricks durch <RUN/STOP RESTORE>, einen Reset oder den Assemblerbefehl BRK rückgängig gemacht werden.

10.1.2 Beschleunigungen des Systems in Basic

Hier erfahren Sie, wie sich die Systembeschleunigungen von Basic aus verwerten lassen. Die Nebenwirkungen bleiben allerdings (siehe 10.1.1).

a. Interrupt einschränken

`POKE 788,52`	verkürzt die Interrupt-Routine um das Abfragen der RUN/STOP-Taste und das Stellen von TI$.
`POKE 788,49`	Normalzustand

In Basic ist das Ausfallen von <RUN/STOP> weitaus tiefgreifender als in Maschinensprache. Überprüfen Sie Ihre Programme deshalb auf Verwendung von TI$ und fügen Sie den POKE erst nach (!) der Fertigstellung des Programms ein.

b. Interrupt abschalten

IRQ aus:	`POKE 56334,PEEK(56334) AND 254`
IRQ an:	`POKE 56334,PEEK(56334) OR 1`

c. Anzahl der Interrupt-Aufrufe ändern

`POKE 56325,0:`	Extrem viele Interruptaufrufe
`POKE 56325,255:`	Extrem wenige (daraus folgt: Interrupt langsam, Basic-Programm schnell)

d. Bildschirm abschalten

Ausschalten:	`POKE 53265,PEEK(53265) AND 239`
Einschalten:	`POKE 53265,PEEK(53265) OR 16`

e. FAST-Modus des C128

Einschalten:	`POKE 53296,1`
Ausschalten:	`POKE 53296,0`

Anhand von Listing 10.1 wollen wir uns nun mit der Anwendung der Systembeschleunigungen befassen.

Listing 10.1: Basic-Testprogramm

```
90 GOTO 200
100 REM >> UP - SCHLEIFE <<
110

120 PRINT" <TASTE>";
    WAIT 198,1
    POKE198,0
    FORI=1TO7
    PRINTCHR$(20);
    NEXT
130

140 FOR I=1 TO 100
    NEXT
150 TI$="000000"
    FOR I = 0 TO 1500
    POKE 53280,I AND 15
    NEXT
    PRINTTI$;TI
    RETURN
160

170 REM >> UP - CURSORBLINKEN AUS <<
180

190 POKE 207,0
    POKE 204,1
    PRINT" "
    RETURN
200 REM -------------------
210 REM -- HAUPTPROGRAMM --
220 REM -------------------
230

240 PRINTCHR$(147)"DEMO FUER SYSTEMBESCHLEUNIGUNGEN (BASIC)";
250 PRINT"----------------------------------------"
260 PRINT"(DOWN)(DOWN)1) NORMALZUSTAND";
    GOSUB100
270

280 PRINT"(DOWN)2) VERKUERZTER INTERRUPT";
    POKE788,52
    GOSUB 100
    POKE 788,49
290

300 PRINT"(DOWN)3) HAEUFIGE INTERRUPTS";
    POKE 56325,20
    POKE204,0
    GOSUB100
    GOSUB170
310

320 PRINT"4) SELTENE  INTERRUPTS";
    POKE 56325,150
    POKE204,0
    GOSUB100
    GOSUB170
330 SYS64931
    REM NORMALZUSTAND EIN
340
350 PRINT"5) BILDSCHIRM ABGESCHALTET ";
    POKE53265,PEEK(53265) AND 239
```

```
    GOSUB140
360 POKE 53265,PEEK(53265) OR 16
    PRINT"(DOWN)** ENDE **"
```

Dieses kleine Beispielprogramm, an welchem Sie nach Herzenslust experimentieren können, unternimmt den Versuch, mit Hilfe von TI$ die Arbeitsdauer der Schleife (Zeile 150) zu messen. Dabei wird bei 5 Testläufen ein und derselben Meßzeile unter veränderten Umständen jeweils die genommene Zeit als TI$- und als TI-Inhalt angezeigt. Während des Ablaufs dieser Schleife, die kontinuierlich die Rahmenfarbe ändert, sollten Sie keine Taste drücken, weil dies die Meßergebnisse verfälschen könnte.
Beachten Sie dies, so erhalten Sie folgende Werte:

1. Normalzustand: 000017 / 1067
2. Verkürzter Interrupt: 000000 / 0 An der gemessenen Zeit können Sie erkennen, daß TI$ abgeschaltet wurde. Gleichzeitig ist <RUN/STOP> während dieses Durchlaufes nicht aktiv.
3. Häufige Interrupts: 000058 / 3515 Aufgrund vieler Interrupt-Anforderungen wurde die Uhr TI$ sehr oft erhöht.
4. Seltene Interrupts: 000007 / 452 Da die IRQ-Routine nur selten durchlaufen wurde, ist TI$ kaum weitergezählt worden.
5. Bildschirm abgeschaltet: 000016 / 1009

Nur bei diesem Punkt (und natürlich auch bei »1«) hat TI$ volle Aussagekraft bezüglich der Ablaufzeit. An dieser Zeit können wir erkennen, daß durch das Abschalten des Bildschirms tatsächlich gegenüber »1« ein Zeitgewinn anfällt. Bei den Punkten »3« und »4« wurde der Cursor eingeschaltet. Bei »3« (häufige Interrupts) ist er sehr schnell, bei »4« dagegen sehr langsam. An Punkt »5« läßt sich auch erkennen, daß bei abgeschaltetem Bildschirm der Hintergrund immer die Rahmenfarbe ($D020) annimmt, ohne daß wir die entsprechende Farbe ins Register $D021 schreiben müssen.

10.2 Optimierung der Bildschirmausgabe

Ohne die Bildschirmausgabe kommt kein Programm aus, aber oft kostet sie unnötig viel Rechenzeit. Der Grund ist hier nicht beim Betriebssystem zu suchen, sondern bei umständlicher Programmierung. Diese wiederum ist auf mangelndes Know-how zurückzuführen, was sich nun ändern wird.

In der Regel wird zur Ausgabe eines Zeichens dieses in den Akku geladen und BSOUT ($FFD2) aufgerufen. Das Betriebssystem prüft dann, ob es sich um ein druckendes Zeichen oder einen Steuercode handelt (siehe »C64 für Insider«). Buchstaben werden in den Bildschirmspeicher geschrieben, für Steuerzeichen existieren jeweils Unterroutinen, die zum Beispiel eine Leerzeile einfügen, den Bildschirm löschen oder ähnliches durchführen.
Diese aufwendige Überprüfung verlangsamt die Bildschirmausgabe. BSOUT ist zwar geringfügig zu beschleunigen, indem man statt bei $FFD2 (Kerneleinsprung) bei $E716 einsteigt (Ausgabe nur auf Bildschirm, nicht auf andere Geräte), aber es geht noch schneller:

a. Bildschirm löschen

Weniger effektiv:
```
LDA #$93 ; $93 = 147 = Code für »Bildschirm löschen«
JSR $FFD2 ; oder JSR $E716
```

Besser:
```
JSR $E544 ; Routine für »Bildschirm löschen«
```

b. Cursor in HOME-Position (linke obere Ecke)

Weniger effektiv:
```
LDA #$13 ; $13 = 19 = Code für »Cursor HOME«
JSR $FFD2 ; oder JSR $E716
```

Besser:
```
JSR $E566 ; Routine für »Cursor HOME«
```

c. Cursor-Positionierung

Weniger effektiv: Senden von Steuerzeichen (CRSR DOWN, UP usw.) über BSOUT

Besser:
```
LDX #zeile
LDY #spalte
JSR $E50C ; Cursorposition setzen
```

d. Textausgabe

Listing 10.2: Zwei Möglichkeiten, einen Text auszugeben

```
100  -;
110  -; TEXTAUSGABE (UEBER BASOUT)
120  -;
130  -.BA $C000 ; START: SYS 49152
140  -;
150  -.GL BASOUT = $FFD2
160  -;
170  -           LDX #0
180  -SCHLEIFE   LDA TEXT,X     ; ZEICHEN LESEN
190  -           INX            ; OFFSET ERHOEHEN
200  -           JSR BASOUT     ; UND AUSGEBEN
```

```
210  -          CPX #18         ; VERGLEICH MIT ANZAHL DER ZEICHEN
220  -          BNE SCHLEIFE    ; SCHON ENDE?
500  -;
510  -; TEXTAUSGABE (UEBER STROUT)
520  -;
530  -.GL STROUT = $AB24
540  -;
550  -          LDA #<(TEXT)    ; LOW-BYTE IN AKKU
560  -          LDY #>(TEXT)    ; HIGH-BYTE IN Y
570  -          STA $22         ; ADRESSE
580  -          STY $23         ; SETZEN
590  -          LDA #18         ; ANZAHL DER ZEICHEN LADEN
600  -          JMP STROUT      ; TEXTAUSGABE UND ENDE
1000 -;
1010 -TEXT      .TX "DAS IST DER TEXT!"
1020 -          .BY 13          ; CARRIAGE RETURN
```

Unkomfortabel:
Senden von Zeichen (Buchstaben, Grafikzeichen) über BSOUT. Eine solche Schleife finden Sie in Listing 10.2, Zeilen 150–220. Nach dem Start durch SYS49152 wird zweimal hintereinander der Text »DAS IST DER TEXT« ausgegeben; das erste Mal wird der Text über eine BSOUT-Schleife gedruckt, das zweite Mal nimmt das Programm die neue Technik.

Komfortabel:
Ab der Adresse TEXT muß sich ein Text als ASCII-Code befinden. Dann wird in $22/$23 diese Adresse TEXT geschrieben sowie in den Akkumulator die Anzahl der auszugebenden Zeichen geladen. Anschließend bewirkt »JSR $AB24« die Textausgabe über eine BSOUT-Schleife. Die Routine $AB24 wird fortan als »STROUT« (STRing OUTput = String-Ausgabe) bezeichnet. STROUT ist zwar langsamer als eine sehr geschickte, eigene BSOUT-Schleife; dafür erlaubt die bequeme Parameterübergabe eine wesentlich einfachere Programmierung, wie Sie am zweiten Teil von Listing 10.2, den Zeilen 500–600, sehen können. Vor allem erhöht die Arbeit mit dem Zeiger $22/$23 die Flexibilität.
Doch es gibt auch einen anderen STROUT-Einsprung bei $AB1E, dessen Anwendung Sie unter anderem an Listing 10.3 sehen können.

Listing 10.3: Anwendung aller vorgestellten Routinen zur Bildschirmausgabe

```
100  -;
110  -; TEXTAUSGABE (UEBER STROUT)
120  -;
130  -.BA $C000 ; START: SYS 49152
140  -;
150  -.GL STROUT = $AB1E
160  -.GL CURSOR = $E50C
170  -.GL CLRSCR = $E544 ; BILDSCHIRM LOESCHEN
180  -;
```

```
190  -.GL ZEILE  = 12
200  -.GL SPALTE = 10
210  -;
300  -          JSR CLRSCR     ; = PRINT CHR$(147)
400  -          LDX #ZEILE     ; ZEILE IN X
410  -          LDY #SPALTE    ; SPALTE IN Y
420  -          JSR CURSOR     ; CURSOR SETZEN
500  -          LDA #<(TEXT)   ; LOW-BYTE IN AKKU
510  -          LDY #>(TEXT)   ; HIGH-BYTE IN Y
520  -          JMP STROUT     ; TEXTAUSGABE & ENDE
600  -;
610  -TEXT      .TX "DAS IST DER TEXT!"
620  -.BY 0 ; ENDMARKIERUNG FUER STROUT
```

Der Einsprung bei $AB1E hat Vor- und Nachteile: Günstig ist, daß nur noch in A/Y die Text-Adresse anzugeben ist; dabei muß am Ende des Textes ein Nullbyte stehen. Sehr ungünstig ist, daß manchmal Probleme bei Eingriffen in den Basic-Interpreter auftreten. Dem $AB24-Einsprung ist also eindeutig Vorzug zu geben; das Hauptargument: Bei $AB24-Aufruf ist die Arbeitsgeschwindigkeit erheblich höher als bei $AB1E.

e. Kopieren des Textes in den Bildschirmspeicher

Dies ist die schnellste Methode: Der fertige Text wird in den Bildschirmspeicher kopiert. Die lange Umwandlung entfällt völlig, da der Text als fertiger Bildschirmcode im Speicher abgelegt wird. Eines müssen Sie aber unbedingt beachten: Die Farbgebung ist nur durch Ändern des Farb-RAMs möglich.
Ein Beispiel ist Listing 10.4.

Listing 10.4: Die schnellste Lösung, einen Text auszugeben

```
100  -;
110  -; TEXT IN VIDEO-RAM SCHREIBEN
120  -;
130  -.BA $C000 ; START: SYS 49152
140  -;
150  -.GL CLRSCR = $E544 ; BILDSCHIRM LOESCHEN
160  -;
170  -.GL ZEILE  = 12
180  -.GL SPALTE = 10
190  -;
200  -.GL VIDEORAM = 1024 ; BILDSCHIRMSPEICHER
210  -.GL ADRESSE = VIDEORAM + (40*ZEILE) + SPALTE
220  -;
300  -          JSR CLRSCR     ; = PRINT CHR$(147)
310  -          LDX #0
320  -SCHLEIFE  LDA TEXT,X     ; BILDSCHIRMCODE LESEN
330  -          BEQ ENDE       ; =0, DANN ENDE
340  -          STA ADRESSE,X  ; IN BILDSCHIRMSPEICHER
350  -          INX
```

```
360  -          JMP SCHLEIFE   ; NAECHSTES ZEICHEN
370  -ENDE      RTS
380  -;
500  -TEXT       .BY 4,1,19," ",9,19,20," "
510  -.BY 4,5,18," ",20,5,24,20,"!"
520  -.BY 0 ; ENDMARKIERUNG DES TEXTES
```

Dieses Programm entspricht in der Wirkung Listing 10.3, gibt den Text jedoch nicht über Kernelroutinen aus, sondern schreibt ihn direkt in den Bildschirmspeicher. In den Zeilen 500–520 steht der Bildschirmcode des Textes.

f. Druckerbetrieb mit STROUT

Folgendes Programm zeigt, wie man zum einen für den STROUT-Einsprung bei $AB1E ein Makro definiert und zum anderen die Ausgabe auf dem Drucker bewältigt:

Listing 10.5: So gibt man einen Text auf dem Drucker aus

```
100  -;
110  -; DRUCKER-AUSGABE MIT
120  -; DER STROUT-ROUTINE
130  -;
140  -.GL STROUT = $AB1E
150  -.GL SETNAM = $FFBD
160  -.GL SETLFS = $FFBA
170  -.GL OPEN   = $FFC0
180  -.GL CHKOUT = $FFC9
190  -.GL CLRCHN = $FFCC
200  -.GL CLOSE  = $FFC3
210  -;
300  -.MA PRINT (ADRESSE)
310  -          LDA #<(ADRESSE)
320  -          LDY #>(ADRESSE)
330  -          JSR STROUT
340  -.RT
350  -;
400  -.BA $C000 ; START: SYS 49152
410  -;
500  -          LDA #0        ; KEINEN
510  -          JSR SETNAM    ; FILENAMEN
520  -;
600  -          LDA #4        ; LOG. FILENUMMER =4
610  -          TAX           ; GERAETEADRESSE 4
620  -          LDY #0        ; SEKUNDAERADRESSE 0
630  -          JSR SETLFS    ; PARAMETER SETZEN
640  -;
700  -          JSR OPEN      ; FILE OEFFNEN
710  -;
800  -          LDX #4        ; FILENUMMER 4
810  -          JSR CHKOUT    ; AUSGABE AUF DRUCKER LENKEN
820  -;
900  -...PRINT (TEXT) ; TEXT AUSGEBEN
910  -;
920  -          JSR CLRCHN    ; WIEDER BILDSCHIRMAUSGABE
930  -;
```

```
940  -...PRINT (TEXT) ; JETZT AUF BILDSCHIRM
950  -;
960  -           LDA #4        ; LOG. FILENUMMER 4
970  -           JMP CLOSE     ; FILE SCHLIESSEN
980  -; & PROGRAMM BEENDEN
990  -;
1000 -TEXT       .TX "DIESER TEXT WIRD AUF"
1010 -.TX " DEN DRUCKER AUSGEGEBEN !"
1020 -.BY 13,13,13,0 ; 3 * CAR.RETURN
```

Es gibt einen Text zuerst auf den Bildschirm und dann auf dem Drucker aus; ist der Drucker ausgeschaltet, so erfolgt zweimal die Bildschirmausgabe.

10.3 Unterprogramme

Ohne die Unterprogramm-Befehle JSR und RTS kommt fast kein Maschinenprogramm aus. Es ist allerdings ziemlich unbekannt, daß beide Befehle das Programm stark verlangsamen. Grund genug für uns, JSR und RTS näher zu betrachten:
Trifft der Prozessor auf JSR, schiebt er den aktuellen Programmzähler plus 2 (= Rücksprungadresse-1) auf den Stack und springt dann zu der Adresse, die hinter JSR steht. Trifft er auf RTS, holt er die Adresse vom Stapel zurück, erhöht sie um 1 und verwendet sie wieder als Programmzähler.
Bemerkenswert ist, daß die Zugriffe auf den Stapel sich in keiner Weise von den Zugriffen über die Befehle PHA und PLA unterscheiden. Daher muß jedesmal der Stapelzeiger neu errechnet werden. Diese vielen Operationen sind schuld daran, daß JSR und RTS so langsam sind. Da wir das Problem erkannt haben, können wir damit beginnen, unser Wissen anzuwenden.

10.3.1 Unterprogramm-Verschachtelung

Stellen wir uns folgendes Beispiel vor: Ein Hauptprogramm ruft das Unterprogramm 1 auf; dieses ruft an seinem Ende das Unterprogramm 2 auf, um dann mit RTS ins Hauptprogramm zurückzukehren. Alles ziemlich schwierig, oder? Deshalb gehen wir mit Hilfe einer Grafik vor: In Bild 10.1 sehen Sie ein Flußdiagramm nach genanntem Aufbau.
In der Beschriftung soll »Code« nicht »Kennwort« bedeuten, sondern heißt einfach »Befehlsmenge«. Wie an den Pfeilen zu erkennen ist, werden zwei RTS-Befehle hintereinander abgearbeitet (von Unterprogramm 2 nach Unterprogramm 1 und von dort wiederum ins Hauptprogramm). Dies ist immer ein Indiz dafür, daß das Programm noch optimiert werden kann. Eine »Übersetzung« von Bild 10.1 in Assembler ist Listing 10.6.

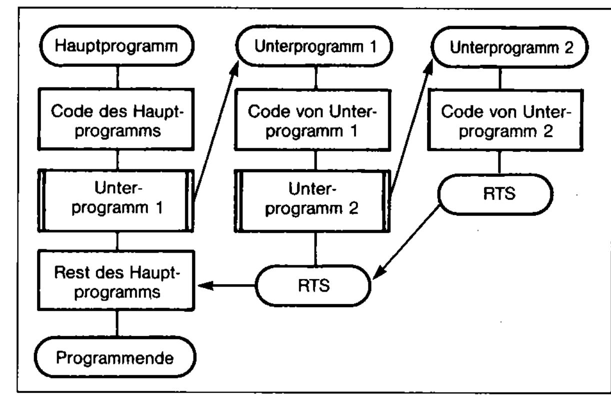


Bild 10.1: Der Algorithmus zur Verschachtelung von Unterprogrammen

Listing 10.6: Die umständliche Methode, Unterroutinen aufzurufen

```
100  -.BA $C000 ; START: SYS 49152
110  -;
120  -; UNTERPROGRAMMVERSCHACHTELUNG IN ASSEMBLER
130  -;
140  -.GL STROUT = $AB1E
150  -;
160  -.MA PRINT (ADRESSE)
170  -           LDA #<(ADRESSE)
180  -           LDY #>(ADRESSE)
190  -           JSR STROUT
200  -.RT
300  -;
310  -; --------------- HAUPTPROGRAMM
320  -;
330  -...PRINT (TEXT1)
340  -;
350  -           JSR UP1
360  -;          ↑ AUFRUF VON UNTERPROGRAMM 1
370  -;
380  -...PRINT (TEXT2)
390  -;
400  -           JMP $A474      ; WARMSTART
410  -;
1000 -;
1010 -; --------------- UNTERPROGRAMM 1
1020 -;
1030 -UP1        NOP            ; BELIEBIGER CODE
1040 -...PRINT (TEXT3)
1050 -;
1060 -           JSR UP2
1070 -;          ↑ AUFRUF VON UNTERPROGRAMM 2
1080 -;
1090 -           RTS            ; UP1 VERLASSEN
1100 -;
2000 -;
2010 -; --------------- UNTERPROGRAMM 2
2020 -;
2030 -UP2        NOP            ; BELIEBIGER CODE
2040 -...PRINT (TEXT4)
2050 -;
2060 -           RTS            ; UP2 VERLASSEN
2070 -;
```

```
5000 -;
5010 -; -------------- TEXTE
5020 -;
5030 -TEXT1      .TX "HIER IST DAS HAUPTPROGRAMM."
5040 -.BY 13,13 ; 1 LEERZEILE
5050 -.BY 0 ; ENDMARKIERUNG
5060 -;
5070 -TEXT2      .TX "HIER IST WIEDER DAS HAUPTPROGRAMM."
5080 -.BY 13,13,0
5090 -;
5100 -TEXT3      .TX "HIER IST DAS UNTERPROGRAMM 1."
5110 -.BY 13,13,0
5120 -;
5130 -TEXT4      .TX "HIER IST DAS UNTERPROGRAMM 2."
5140 -.BY 13,13,0
```

Wenn Sie dieses Programm über SYS49152 starten, ist aus den ausgegebenen Texten ersichtlich, welcher Programmteil zu welchem Zeitpunkt abgearbeitet wird. Sobald Sie die Struktur von Bild 10.1 beziehungsweise Listing 10.6 verstanden haben, können wir uns mit der optimierten Form befassen, die in Bild 10.2 und Listing 10.7 zu finden ist.

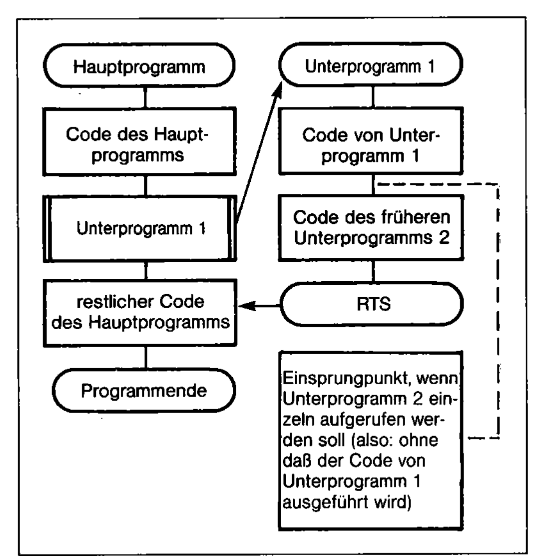


Bild 10.2: Der optimierte Algorithmus zur Verschachtelung von Unterprogrammen

Listing 10.7: Die optimierte Methode, Unterroutinen aufzurufen

```
100  -.BA $C000 ; START: SYS 49152
110  -;
120  -; UNTERPROGRAMMVERSCHACHTELUNG
130  -; (OPTIMIERTE ASSEMBLERVERSION)
140  -;
150  -.GL STROUT = $AB1E
160  -;
170  -.MA PRINT (ADRESSE)
```

```
180  -            LDA #<(ADRESSE)
190  -            LDY #>(ADRESSE)
200  -            JSR STROUT
210  -.RT
500  -;
510  -; --------------- HAUPTPROGRAMM
520  -;
530  -...PRINT (TEXT1)
540  -;
550  -            JSR UP1
560  -;           ↑ AUFRUF VON UNTERPROGRAMM 1
570  -;
580  -...PRINT (TEXT2)
590  -;
600  -            JMP $A474       ; WARMSTART
610  -;
1000 -;
1010 -; --------------- UNTERPROGRAMM 1
1020 -;
1030 -UP1        .BA UP1         ; BELIEBIGER CODE
1040 -...PRINT (TEXT3)
1050 -;
1060 -;
2000 -;
2010 -; --------------- CODE VON UNTERPROGRAMM 2
2020 -;
2030 -UP2        .BA UP2         ; BELIEBIGER CODE
2040 -           LDA #<(TEXT4)   ; LOW-BYTE
2050 -           LDY #>(TEXT4)   ; HIGH-BYTE
2060 -           JMP STROUT      ; TEXTAUSGABE
2070 -; UND RUECKSPRUNG VOM UNTERPROGRAMM,
2080 -; WEIL AM ENDE DER STROUT-ROUTINE
2090 -; EIN RTS-BEFEHL STEHT.
2100 -;
10000-;
10010-; --------------- TEXTE
10020-;
10030-TEXT1      .TX "HIER IST DAS HAUPTPROGRAMM."
10040-.BY 13,13 ; 1 LEERZEILE
10050-.BY 0 ; ENDMARKIERUNG
10060-;
10070-TEXT2      .TX "HIER IST WIEDER DAS HAUPTPROGRAMM."
10080-.BY 13,13,0
10090-;
10100-TEXT3      .TX "HIER IST DAS UNTERPROGRAMM 1."
10110-.BY 13,13,0
10120-;
10130-TEXT4      .TX "HIER IST DAS UNTERPROGRAMM 2."
10140-.BY 13,13,0
```

Hier wird das ehemalige Unterprogramm 2 ans Ende von Unterprogramm 1 gehängt (wobei es ebenfalls über JMP UP2 angesprungen werden könnte). Auf diese Weise muß es nicht über JSR aufgerufen werden, was einen RTS-Befehl überflüssig macht. Trotz dieser Änderung kann das Unterprogramm 2 auch weiterhin als Unterprogramm aufgerufen werden, da bei JSR UP2 die CPU auf einen RTS-Befehl trifft (Bild 2).

In Listing 10.7 ist noch der JMP-Befehl in Zeile 2060 erläuterungsbedürftig: Dort muß nicht »JSR STROUT:RTS« stehen, weil am Ende der STROUT-Routine im ROM ohenhin ein RTS steht. Deshalb benötigt unser Programm keinen eigenen RTS-Befehl zur Rückkehr ins Hauptprogramm. Die folgende Regel gilt für Aufrufe von Betriebssystemroutinen:

```
JSR $XXXX          entspricht     JMP $XXXX
RTS
```

Voraussetzung ist, daß im Unterprogramm ab $XXXX keine Stapelmanipulation erfolgt, wie sie gleich beschrieben wird. Das geschilderte Verfahren zur Unterprogrammverschachtelung und die entsprechenden Regeln können Sie dann auf jede (!) Programmiersprache übertragen.

10.3.2 Stapelmanipulation

Wenn Sie »Exbasic Level II« kennen, wissen Sie sicher den Befehl »DISPOSE RETURN« zu schätzen. Er dient dazu, ein Unterprogramm ohne RETURN abzuschließen. Dadurch kann dieses zum Beispiel über GOTO verlassen werden. In Assembler ist dies auch möglich. Die Befehlsfolge

```
PLA
PLA
```

entspricht in der Wirkung »DISPOSE RETURN«. Da die Rücksprungadresse auf den Stapel gelegt wird und dort genau 2 Byte in Anspruch nimmt, kann sie über PLA:PLA wieder vom Stapel geholt werden. Ein Unterprogramm ist nach PLA:PLA eigentlich kein Unterprogramm mehr, sondern Bestandteil des aufrufenden Programms. PLA:PLA findet vor allem in der Fehlerbehandlung Anwendung.
Nach PLA:PLA kann ein Unterprogramm über JMP verlassen werden, so wie wenn es auch nicht über JSR, sondern über JMP aufgerufen worden wäre. Dies machen wir uns zunutze, um den Rücksprung an eine beliebige Adresse zu simulieren. Dies ist sonst nicht möglich, da bei RTS immer hinter den Befehl gesprungen wird, der das Unterprogramm aufgerufen hat. Ein RTS an eine beliebige Adresse müßte »RTS XXXX« heißen, doch diesen Befehl gibt es beim 6510 nicht. So wird er aber simuliert:

```
PLA       ; holt Rücksprungadresse
PLA       ; vom Stapel und
JMP $XXXX ; springt nach $XXXX
```

So sieht ein Makro dazu aus:

```
-.MA      RTS (RUECKSPRUNGADRESSE)
-         PLA
-         PLA
-         JMP RUECKSPRUNGADRESSE
-.RT
```

Und noch ein Mangel der Unterprogrammbefehle soll nun beseitigt werden: Obwohl es JMP(indirekt) gibt, kennt der 6510 keinen Befehl wie JSR(indirekt); über Stapelmanipulation ist dies dennoch möglich. Nehmen wir an, der Vektor \$14/\$15 enthält die Adresse \$C000. Nun soll über den Vektor \$14/\$15 ein Unterprogramm aufgerufen werden (also das ab \$C000, da diese Adresse im Vektor steht). Bild 10.3 zeigt, wie dies geschehen muß.

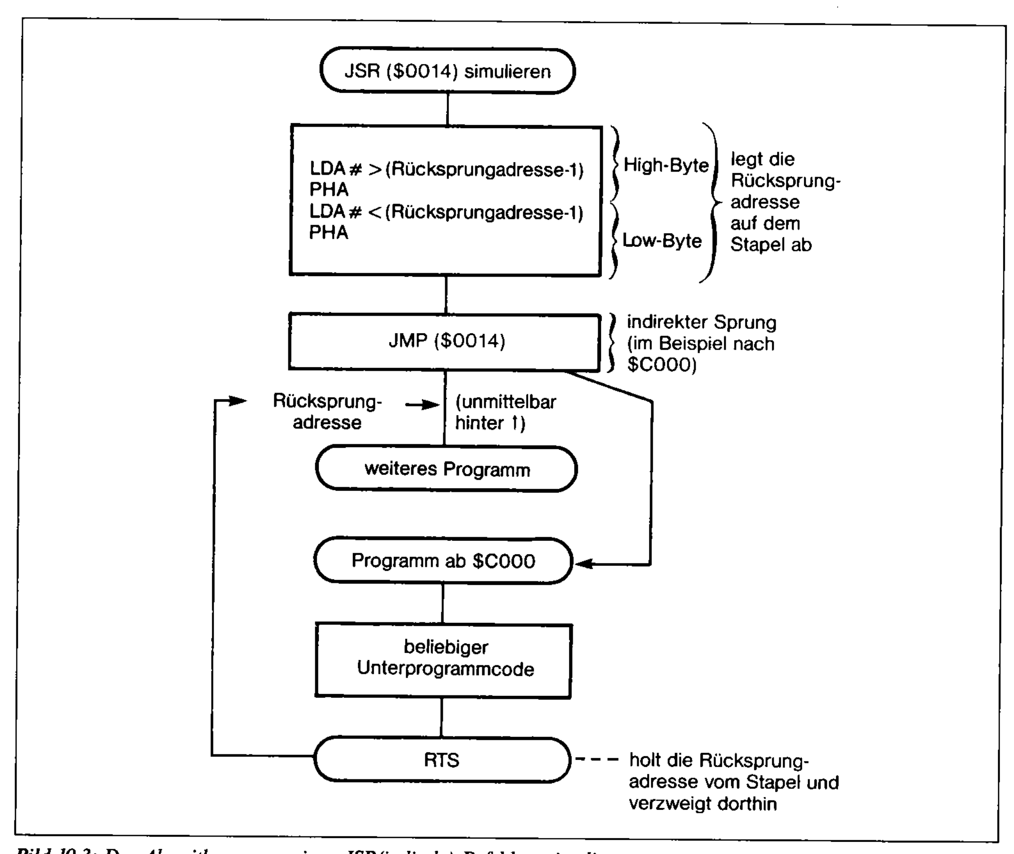


Bild 10.3: Der Algorithmus, um einen JSR(indirekt)-Befehl zu simulieren

Die Rücksprungadresse befindet sich zwar in Bild 10.3 direkt hinter dem JMP(\$0014)-Befehl, kann aber auch anderswo im Programm liegen. Folgendes Makro ermöglicht die Simulation von JSR(indirekt):

```
-.MA JSRIND (VEKTOR,RUECKSPRUNGADRESSE)
-          LDA #>(RUECKSPRUNGADRESSE-1)
-          PHA
```

```
-          LDA #<(RUECKSPRUNGADRESSE-1)
-          PHA
-          JMP (VEKTOR)
-.RT
```

Diese Simulation von JSR($XXXX) verendet auch der SYS-Befehl (siehe »C 64 für Insider«). Später werden wir noch eine weitere Möglichkeit für JSR(ind) kennenlernen, die aber nicht auf Stapelmanipulation beruht.

10.4 Tabellen

Im allgemeinen Sprachgebrauch werden Tabellen als »geordnete Zusammenstellungen von Daten« verstanden. Diese Funktion üben sie auch in Computerprogrammen aus, wo man sie daran erkennt, daß Tabellen keinen Befehlscharakter haben.
Wozu werden nun Tabellen verwendet? In der Regel dienen Tabellen einem Computerprogramm als »elektronischer Rechenschieber«. So wie das Kopfrechnen durch einen Rechenschieber ersetzt werden kann, weil man nur in einer geordneten Zusammenstellung von Ergebnissen das richtige suchen muß, kann ein Programm aus seinen Tabellen denselben Nutzen ziehen: Die Berechnungen entfallen, die Programmierung wird einfacher.
Aus den wenigen erforderlichen Rechnungen entsteht ein deutlicher Geschwindigkeitszuwachs, der Hauptvorteil von Tabellen. Wie man Tabellen einsetzt, erfahren Sie im folgenden.

10.4.1 Tabellen aus Rechenergebnissen

Noch einmal zum Rechenschieber. Es geht beim Kopfrechnen viel schneller, 14 x 10 auszurechnen als 14 x 7. Bei einem Rechenschieber besteht kein Unterschied in der »Rechenzeit« (richtig: »Tabellen-Auswertungszeit«).
Dementsprechend existiert fast kein Algorithmus, dessen Ausführungszeit bei unterschiedlichen Parametern immer gleich bliebe. Ersetzt, beziehungsweise unterstützt man einen Algorithmus durch eine Multiplikationstabelle, fällt nicht nur eine kürzere, sondern auch eine einheitlichere Ausführungszeit an. Für das Rechnen mit einzelnen Bits in einem Byte werden oft die Zweierpotenzen benötigt; es empfiehlt sich, diese als Tabelle anzulegen:

```
1000 -; Zweierpotenzen als Tabelle
1010 -; im DOS der Floppy 1541 ab $EFE9 zu finden
1020 -;
1030 -ZWEIPOT .BY 1,2,4,8,16,32,64,128
```

Folgende Unterroutine legt im Akkumulator den Wert »2 hoch A« ab, wobei unter A der Inhalt des Akkumulators bei Aufruf der Routine zu verstehen ist:

```
10000 -;
10010 -; Unterroutine zur Berechnung von
10020 -; 2↑ A (2 hoch A); Ergebnis kommt in Akku
10030 -;
10040 -  TAX ; Akku in Indexregister
10050 -  LDA ZWEIPOT,X ; aus Tabelle auslesen
10060 -  RTS ; das war's schon!
10070 -;
10080 -  ZWEIPOT .BY 1,2,4,8,16,32,64,128
```

Wenn A größer als 7 ist, liefert das Programm falsche Werte. Sie können es aber noch erweitern, wenn Sie es für nötig halten.

10.4.2 Tabellen aus Fließkommawerten

Zu den zeitraubendsten Operationen gehört die Rechnung mit Fließkommazahlen. Daß diese selbst in Maschinenprogrammen lähmend wirkt, wissen Sie bestimmt aus eigener leidvoller Erfahrung. Daher sollte man nur dann auf die Fließkomma-Routinen zugreifen, wenn es unvermeidbar ist. Berechnen Sie so viele Werte wie möglich voraus, hierfür eignet sich der Direktmodus des Basic-Interpreters besonders gut! Wie Sie einen auf diese Weise berechneten Wert ins MFLPT-Format (siehe »C 64 für Insider«) umwandeln können, zeigt Ihnen folgende Anleitung:

Verfahren zur Umwandlung einer Zahl ins MFLPT-Format

1. SMON oder anderen Monitor laden
2. Reset auslösen oder NEW eingeben
3. »XX=Fließkommazahl« eingeben, zum Beispiel »XX=1.23456«
4. Monitor starten (z.B. SYS49152)
5. »M 0805 0809« eingeben

Sie sehen nun in den Adressen $0805–$0809 die MFLPT-Darstellung der Zahl, mit der Sie die Variable XX belegt haben.
Damit wir uns unter Zuhilfenahme präziser Fachausdrücke und Abkürzungen verständigen können, sollten Sie bereits Grundkenntnisse über das Fließkommaformat erworben haben. In »C 64 für Insider« vermittelt ein eigener Abschnitt den Einstieg in die Welt der Fließkommazahlen, der anhand zahlreicher Beispiele auch die trockene Theorie anschaulich darbieten möchte. Sehr wichtig ist auch, daß man nicht nur mit allen ROM-Routinen zur Fließkommarechnung umgehen kann, sondern auch deren Wirkungsweise erfährt; leider gehen bisherige ROM-Listings darauf in recht hilfloser Weise ein, weshalb ich beim Verfassen von »C 64 für Insider« besonderen Wert darauf gelegt habe, diese Zusammenhänge für jeden verständlich zu machen. Deshalb wollte ich auch an dieser Stelle darauf hinweisen.

Zurück zum 5-Schritte-Verfahren. Im Falle der Zahl 1.23456 erhalten wir als Ergebnis

```
:0805  81 1E 06 0F E5 ...
```

Diese Werte legen wir folgendermaßen als Tabelle ab:

```
1040 -BSPZAHL .BY $81, $1E, $06, $0F, $E5
```

Wie wir nun diese Zahl verarbeiten, zeigt Ihnen Listing 10.8.

Listing 10.8: Fließkommazahlen in Assembler verarbeiten

```
100   -.BA $C000 ; START: SYS 49152
110   -;
120   -; RECHNUNG MIT FLIESSKOMMAWERTEN
130   -;
140   -.GL MEMFAC = $BBA2
150   -.GL FACOUT = $AABC
160   -.GL SQRFAC = $BF71
170   -.GL LOGNAT = $B9EA
180   -;
300   -.MA HOLE (ADRESSE) ; MAKRO-DEF.
310   -          LDA #<(ADRESSE); HOLT MFLPT-ZAHL
320   -          LDY #>(ADRESSE); VON ADRESSE IN
330   -          JSR MEMFAC     ; DEN FAC
340   -.RT
350   -;
500   -;
510   -...HOLE (BSPZAHL)
520   -;
530   -          JSR FACOUT     ; AUSDRUCKEN
600   -;
610   -...HOLE (BSPZAHL)
620   -;
630   -          JSR SQRFAC     ; QUADRATWURZEL
640   -;
650   -          JSR FACOUT     ; AUSDRUCKEN
700   -;
710   -...HOLE (BSPZAHL)
720   -;
730   -          JSR LOGNAT     ; LOGARITHMUS NATURALIS
740   -;
750   -          JMP FACOUT     ; AUSDRUCKEN
1000 -;
1010 -; BEISPIELZAHL 1.23456
1020 -; IM MFLPT-FORMAT
1030 -;
1040 -BSPZAHL   .BY $81,$1E,$06,$0F,$E5
1050 -;
```

Das Makro (300–340) stützt sich auf die Interpreter-Routine, die eine Zahl (Adresse in A/Y übergeben) vom Speicherformat MFLPT in den FAC als FLPT-Zahl schreibt und dabei die erforderliche Umwandlung durchführt.

In der Tabelle in Zeile 1040 können Sie beliebige Fließkommawerte (sofern Sie diese, wie angegegeben, berechnet haben) einsetzen, das Programm rechnet dann mit der jeweiligen Beispielzahl, die ab BSPZAHL im MFLPT-Format steht. Diese Zahl wird zunächst nur in den FAC geladen und der FAC dann ausgedruckt, dann wird die Zahl wieder geholt, die Wurzel sowie der natürliche Logarithmus berechnet und auch diese ausgegeben.

10.4.3 Sprungtabellen

Beim Thema »Unterprogramme« wurde Ihnen eine Methode vorgestellt, um JSR(ind) zu simulieren. Diese erweist sich in Verbindung mit einer Tabelle, in der die Sprungadressen gespeichert sind, als überaus nützlich. So kann beispielsweise eine Parallele zum Basic-Befehl ON...GOSUB geschaffen werden.
Ein Beispiel: Wenn der Basic-Interpreter auf einen Basic-Befehl trifft, holt er aus der Tabelle $A00C–$A09D die Adresse der zugehörigen Routine. Diese springt er dann durch Stapelmanipulation an.
Der bekannte Maschinensprache-Monitor SMON, veröffentlicht u.a. im 64'er-Sonderheft 8/85, arbeitet genauso: Seine Sprungtabelle liegt im Bereich $C02B–$C06B. Im folgenden werde ich des öfteren auf den SMON zurückgreifen, da dieser nicht nur ein populäres, sondern eben auch lehrreiches Programm ist. Ich beziehe mich dabei immer auf die Version, die im 64'er-Magazin abgedruckt wurde.
Im folgenden lernen Sie die Anwendung an mehreren Stellen kennen, denn Sprungtabellen sind für sich allein genommen kaum einzusetzen. Vielmehr treten sie im direkten Zusammenhang mit Vergleichstabellen auf.

10.4.4 Vergleichstabellen

Weder der SMON noch der Basic-Interpreter benutzen zum Suchen der zum jeweiligen Befehl gehörenden Routine eine Reihe von CMP-Abfragen mit BRANCH-Befehlen. Auch für die Vergleichswerte (in diesem Fall die Befehlswörter) gibt es eine Tabelle: Beim SMON liegt sie im Bereich $C00B–$C02A, beim Basic-Interpreter $A09E–$A19D.
Sprung- und Vergleichstabellen sind in gleicher Befehlsfolge angeordnet; wird der Befehl an einer bestimmten Stelle in der Vergleichstabelle gefunden, erfolgt ein Sprung an die Adresse, die an gleicher Position in der Sprungtabelle steht. So sehen die Befehls- und Vergleichstabellen im SMON aus:

Spalte Nr.	1	2	3	4	...
Befehl	'	#	$	%	...
Sprungadr. $	CADB	C92E	C908	C91C	...

Die Sprungadressen sind wegen der Stapelmanipulation in der Tabelle ab $C02B um 1 dekrementiert gespeichert (siehe 10.3.2); in der Darstellung sehen Sie aber das tatsächliche Sprungziel. Wir werden jetzt anhand des SMON die Verwendung einer Vergleichs-Sprungtabelle in Assembler erläutern. Wenn wir die zum Befehl » # « gehörende Sprungadresse fnden wollen, gehen wir folgendermaßen vor:

1. Wir suchen in Reihe 2 das #-Zeichen.
2. Wir gehen (in derselben Spalte) eine Reihe nach unten und finden dort die Sprungadresse ($C92E).

Der Computer hat nicht die Möglichkeit, direkt eine Reihe weiter unten die Suche fortzusetzen. Er muß einen Umweg wählen und sich die Spalte merken. Ein Beispiel:

1. Der SMON sucht unter den Elementen aus Reihe 2 das » # «. In einem Zähler merkt er sich die Spalte, in welcher der Befehl gefunden wurde.
2. Nun sucht er in Reihe 3 in der Spalte, die der Zähler angibt, die zugehörige Sprungadresse.

Wie ähnlich beide Suchvorgänge sind, erkennen Sie daran, daß jedesmal die Hauptschritte 1 und 2 vorkommen. Nach soviel Theorie sehen wir uns nun um so ausführlicher die Routine im SMON an, die für die Steuerung der Vergleichs-Befehlstabelle verantwortlich ist. Dazu können Sie »D C303 C323« eingeben. Bei Adresse $C303 steht im Akku der ASCII-Code des Kommandos, das der SMON ausführen soll (zum Beispiel $4D, wenn ein M-Befehl eingegeben wurde).

`C303`	`LDX #$20`	32–1 Befehle müssen durchsucht werden. Weshalb »–1« erforderlich ist, liegt an der Schleifenstruktur und ist unbedeutend.
`C305`	`CMP $C00A,X`	Akku (enthält Befehl als ASCII-Code) mit x-tem Element der Befehlstabelle vergleichen; $C00A = Befehlstabelle –1, weil Adresse $C00A nie zum Vergleich herangezogen wird.
`C308`	`BEQ $C30F`	Vergleich positiv; im X-Register steht jetzt die Spalte.
`C30A`	`DEX`	Zähler wird dekrementiert; es handelt sich hier um eine »Dekrementierschleife« (dieses Thema wird noch behandelt).
`C308`	`BNE $C305`	Wenn der Zähler noch nicht gleich 0 ist, folgt ein Sprung zum Schleifenbeginn.
`C30D`	`BEQ $C2D1`	Wenn X=0, dann wurde die ganze Tabelle durchsucht, und der Befehl nicht gefunden! Deshalb wird in die SMON-Fehlerbehandlung gesprungen.
`C30F`	`JSR $C315`	Diese Stelle wird von $C308 aus angesprungen; hier wiederum steht ein Aufruf des Unterprogramms ab $C315.

C312	JMP $C2D6	Nachdem nun der Befehl durch die Subroutine $C315 abgearbeitet wurde, folgt ein Sprung zur Eingabe des nächsten Befehls.
C315	TXA	Das ist sie, die Subroutine! Weil im X-Register die Nummer des Befehls (= Spalte in Tabelle) steht, kommt das X-Register ins Hauptrechenregister, den Akkumulator.
C316	ASL	Die Befehlsnummer wird mit 2 multipliziert...
C317	TAX	...und kommt wieder ins X-register.
C318	INX	Das X-Register wird um 1 erhöht, da das High-Byte eine Position hinter dem Low-Byte steht.
C319	LDA $C029,X	High-Byte wird gelesen. Die Sprungtabelle beginnt zwar 2 Byte nach $C029, aber weil es keine Spalte 0 gibt, muß der Speicherbedarf einer Sprungadresse (=2) abgezogen werden.
C31C	PHA	Das High-Byte der Adresse wird auf den Stapel gelegt.
C31D	DEX	–1, weil Low-Byte eine Adresse vor High-Byte steht (Low-High-Format!).
C31E	LDA $C029,X	Nun wird auch das Low-Byte der Adresse...
C321	PHA	...auf den Stapel geschoben.
C322	RTS	Der Befehl RTS wird hier zur Simulation von JMP(ind) verwendet. Auf dieses (unpraktische) Verfahren soll nicht weiter eingegangen werden, weil der 6510 den Befehl JMP(ind) kennt. Wichtig ist jetzt nur noch zu wissen, daß jede Befehlsroutine des SMON mit einem RTS endet, der einen Rücksprung zur Adresse $C312 bewirkt.

10.5 Prüfsummen

Nun lernen wir ein besonders raffiniertes Vergleichsverfahren kennen. Wie gesagt, benötigen Vergleiche mit Wörtern, die aus unterschiedlich vielen Zeichen bestehen, mehr Taktzyklen. Dem wäre nicht so, wenn wir alle Zeichen auf eine einheitliche Länge bringen würden. Genau dies tut der Basic-Interpreter:
Bei Eingabe einer Zeile wandelt er alle Basic-Befehlswörter in Token um (siehe »C 64 für Insider«). Jedes Token vertritt einen Basic-Befehl und kann, da es nur ein Byte benötigt, schneller erkannt werden, als es bei mehreren Bytes möglich wäre. Ein Nachteil ist jedoch der Speicherplatzaufwand; für die Umwandlung müssen die Befehlswörter irgendwo im Speicher in Langform vorhanden sein.

Es gibt aber noch ein anderes Verfahren, einer Zeichenkette einen Wert zuzuweisen: die Prüfsummenberechnung.
Diese führen zum Beispiel die allseits beliebten Eingabehilfen »Checksummer« und »MSE« des 64'er-Magazins durch: Aus 8 Byte Programmcode und 2 Byte Adresse errechnet der MSE eine 1-Byte-Prüfsumme, aus einer Programmzeile von unterschiedlicher Länge der Checksummer ebenfalls 1 Byte als Prüfwert.
In Bild 10.4 sehen Sie das Flußdiagramm für einen sehr zuverlässigen Algorithmus zur Berechnung von Prüfsummen (insofern zuverlässig, als er sehr unterschiedliche Prüfsummen ermittelt). Listing 10.9 stellt ein Hilfsprogramm dar, das zu einer Eingabe die Prüfsumme nach dem Algorithmus aus Bild 10.4 errechnet.

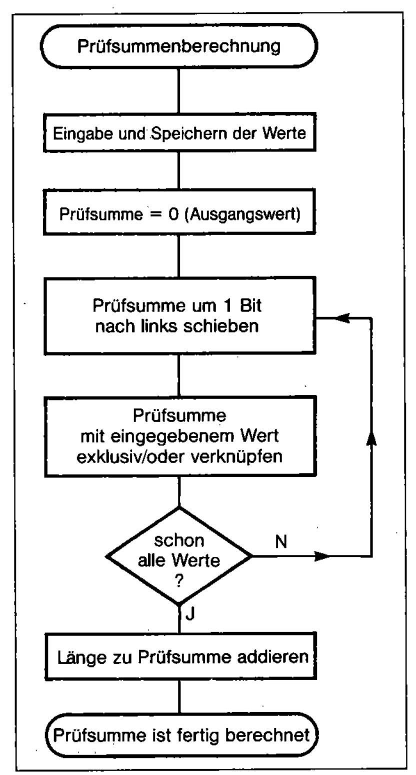


Bild 10.4: Das Flußdiagramm zur Prüfsummenberechnung

Listing 10.9: Die Berechnung von Prüfsummen

```
100  -;
110  -.BA $C000 ; START: SYS 49152
120  -;
130  -.GL BASIN = $FFCF
140  -.GL NUMOUT = $BDCD
150  -.GL STROUT = $AB1E
160  -;
170  -ANFANG     LDA #<(TEXT1)
```

```
180  -            LDY #>(TEXT1)
190  -            JSR STROUT
200  -;
210  -            LDX #0
220  -SCHLEIFE1 JSR BASIN
230  -            CMP #13         ; 13 = RETURN
240  -            BEQ WEITER
250  -            STA STORE,X
260  -            INX
270  -            JMP SCHLEIFE1
280  -;
290  -WEITER     STX LAENGE
300  -            LDA #<(TEXT2)
310  -            LDY #>(TEXT2)
320  -            JSR STROUT
330  -            LDA #0
340  -; 0 = AUSGANGSWERT DER PRUEFSUMME
350  -            TAX             ; ZAEHLER = 0
360  -SCHLEIFE2 ROL              ; PRUEFSUMME * 2
370  -            EOR STORE,X
380  -            INX             ; ZAEHLER ERHOEHEN
390  -            CPX LAENGE
400  -            BNE SCHLEIFE2
410  -            CLC
420  -            ADC LAENGE      ; LAENGE ADDIEREN
430  -            TAX             ; PRUEFSUMME
440  -            LDA #0          ; AUSGEBEN
450  -            JSR NUMOUT
460  -            JMP ANFANG      ; NOCH EINMAL
1000 -;
1010 -; TEXTE
1020 -;
1030 -TEXT1      .BY 13
1040 -.TX "----------------------------------------"
1050 -.TX "EINGABE ?  "
1060 -.BY 0
1070 -;
1080 -TEXT2      .BY 13
1090 -.TX "PRUEFSUMME "
1100 -.BY 0
2000 -;
2010 -; ZWISCHENSPEICHER
2020 -;
2030 -LAENGE     .BY 0           ; ZWISCHENSPEICHER
2040 -STORE      .BY 0
2050 -; ↑ AB STORE WIRD DIE EINGABE ABGELEGT
```

In Listing 10.9 ist Ihnen eventuell die Routine NUMOUT nicht bekannt; daher sei eine kurze Beschreibung gegeben (alle Details finden Sie in »C64 für Insider«): NUMOUT gibt eine positive Integerzahl, deren High-Byte im Akkumulator und Low-Byte im X-Register übergeben wird, aus. NUMOUT wird zum Beispiel von der LIST-Routine bei der Ausgabe einer Zeilennummer aufgerufen.

Die Routine BASIN soll ebenfalls kurz erklärt werden, da sie in allen folgenden Programmen intensiv verwendet wird. Wenn die Routine BASIN zum ersten Mal aufgerufen wird, erwartet das Betriebssystem eine Eingabe (normalerweise von Tastatur), die der Eingabe einer Basic-Zeile entspricht. Nach der Eingabe wird das erste eingegebene Byte in den Akku geladen, jeder weitere Aufruf von BASIN holt das nächste Zeichen in den Akku. Wurden alle Bytes eingelesen, wird im Akku der Wert 13 ($0D, RETURN) übergeben; danach führt ein weiterer BASIN-Aufruf zu erneuter Tastatureingabe.

Zurück zum Thema »Prüfsummen«. Ein großer Vorteil von Prüfsummen ist, daß die Vergleiche mit nur einem Byte, nämlich der Prüfsumme, durchzuführen sind. Wie man in den Genuß dieses Vorteils kommt, zeigt Listing 10.10.

Listing 10.10: Eine Anwendung der Prüfsummenberechnung

```
100  -;
110  -.BA $C000 ; START: SYS 49152
120  -;
130  -.GL BASIN = $FFCF
140  -.GL NUMOUT = $BDCD
150  -.GL STROUT = $AB1E
160  -;
170  -ANFANG    LDA #<(TEXT1)
180  -          LDY #>(TEXT1)
190  -          JSR STROUT
200  -;
210  -          LDX #0
220  -SCHLEIFE1 JSR BASIN
230  -          CMP #" "        ; SPACE?
240  -          BEQ SCHLEIFE1   ; DANN UEBERLESEN
250  -          CMP #13         ; 13 = RETURN
260  -          BEQ WEITER1
270  -          STA STORE,X
280  -          INX
290  -          JMP SCHLEIFE1
300  -;
310  -WEITER1   STX LAENGE
320  -          LDA #<(TEXT2)
330  -          LDY #>(TEXT2)
340  -          JSR STROUT
350  -          LDA #0
360  -; 0 = AUSGANGSWERT DER PRUEFSUMME
370  -          TAX             ; ZAEHLER = 0
380  -SCHLEIFE2 ROL             ; PRUEFSUMME * 2
390  -          EOR STORE,X
400  -          INX             ; ZAEHLER ERHOEHEN
410  -          CPX LAENGE
420  -          BNE SCHLEIFE2
430  -          CLC
440  -          ADC LAENGE      ; LAENGE ADDIEREN
450  -; HIER STEHT DIE PRUEFSUMME IM AKKU
460  -;
470  -          LDX #0
480  -SCHLEIFE3 CMP PRUEFSUMMEN,X
490  -          BEQ WEITER2
500  -          INX
510  -          CPX #4
520  -          BNE SCHLEIFE3
530  -; PRUEFSUMME NICHT GEFUNDEN
540  -;
550  -          PLA
```

```
560  -             PLA
570  -             LDA #<(TEXT3)
580  -             LDY #>(TEXT3)
590  -             JSR STROUT
600  -             JSR ANFANG      ; VON VORNE
610  -;
620  -WEITER2      LDA LOWTAB,X    ; LOW-BYTE
630  -             LDY HIGHTAB,X   ; HIGH-BYTE
640  -             JSR STROUT
650  -             JMP ANFANG      ; NOCH EINMAL!
660  -;
1000 -;
1010 -; TEXTE
1020 -;
1030 -TEXT1        .BY 13
1040 -.TX "--------------------------------------"
1050 -.TX "COMPUTER : "
1060 -.BY 0
1070 -;
1080 -TEXT2        .BY 13
1090 -.TX "PROZESSOR: "
1100 -.BY 0
1110 -;
1120 -TEXT3        .TX "WEISS ICH NICHT!"
1130 -.BY 0
1140 -;
1150 -;
1160 -T6502        .TX "MOS 6502"
1170 -.BY 0
1180 -;
1190 -T6510        .TX "MOS 6510"
1200 -.BY 0
1210 -;
1220 -T8502        .TX "MOS 8502 & Z80"
1230 -.BY 0
1240 -;
1250 -T68000       .TX "MOTOROLA 68000"
1260 -.BY 0
1270 -;
2000 -;
2010 -; NUMERISCHE TABELLEN
2020 -;
2030 -LOWTAB       .BY <(T6502),<(T6510),<(T8502),<(T68000)
2040 -HIGHTAB      .BY >(T6502),>(T6510),>(T8502),>(T68000)
2050 -;
2060 -PRUEFSUMMEN .BY 228,83,149,136
2070 -; REIHENFOLGE: VC20,C64,PC128,AMIGA
3000 -;
3010 -; ZWISCHENSPEICHER
3020 -;
3030 -LAENGE       .BY 0            ; ZWISCHENSPEICHER
3040 -STORE        .BY 0
3050 -; ↑ AB STORE WIRD DIE EINGABE ABGELEGT
```

Wenn Sie den Namen eines Computers (C64, VC 20, PC 128 oder AMIGA) eingeben, nennt das Programm den in diesem Computer installierten Mikroprozessor. Bei der Eingabe der Computernamen kann man aufgrund der Zeilen 230 und 240 beliebig viele Leerzeichen ein-

setzen. Bei der Errechnung der Prüfsummen mit Listing 10.9 dürfen allerdings keine Leerzeichen eingegeben werden, da Listing 10.9 diese nicht überliest und somit ein falsches Ergebnis liefern würde.
Der Programmteil, der die Prüfsumme der Eingabe berechnet, ist mit Ausnahme der Zeilen 230/240 aus Listing 10.9 übernommen worden. Nach Zeile 450 wird die ermittelte Prüfsumme mit der Tabelle PRUEFSUMMEN (Zeile 2060) verglichen. Bei WEITER2 (Zeile 620) steht im X-Register die Spalte, in der die Prüfsumme gefunden wurde. Listing 10.10 numeriert die Spalten mit 0 beginnend; außerdem teilt sich die Adressentabelle in LOWTAB (Tabelle der Low-Bytes) und HIGHTAB (High-Bytes) auf, was die Programmierung stark erleichtert.
Steht also im X-Register die Spalte (0 = VC 20, 1 = C64, 2 = PC 128, 3 = AMIGA), so lesen die Zeilen 620/630 aus der besagten zweiteiligen Tabelle LOWTAB/HIGHTAB die Adresse, ab welcher die ASCII-Darstellung des Prozessortyps zu finden ist.
Einer akuten Gefahr bei der Verwendung von Prüfsummen muß man sich jedoch immer bewußt sein: »Überschneidung« von Prüfsummen. Wenn Sie bei der Anwendung von Listing 10.10 etwas herumexperimentieren, werden Sie sicher feststellen, daß auch eigentlich nicht vorgesehene Eingaben Wirkung zeigen. Dies liegt daran, daß sich die vorgesehenen Eingaben nicht in ihren Prüfsummen überschneiden, d.h. die gleichen Prüfsummen haben. Wenn man dies aber beachtet, so ist das Arbeiten mit Prüfsummen, vor allem bei kleineren Datenmengen, eine angenehme Sache.

10.6 Umfassendes Beispielprogramm für Tabellen

Wenden wir uns nun einem etwas größeren, aber keineswegs komplizierteren Programm zu. Es heißt schlicht und einfach »TABELLEN-BEISPIEL«, womit schon einiges über die Funktion ausgesagt ist: ein reines Beispielprogramm, das nicht den Anspruch erhebt, etwa als Anwendersoftware nützlich zu sein. In Listing 10.11 finden Sie den dokumentierten Quelltext.

Listing 10.11: TABELLEN-BEISPIEL

```
100  -.BA $C000 ; START: SYS 49152
110  -;
120  -; *************************
130  -; *                       *
140  -; *  TABELLEN - BEISPIEL  *
150  -; *  =================    *
160  -; *                       *
170  -; *  BY  FLORIAN MUELLER  *
180  -; *                       *
190  -; *************************
200  -;
210  -.GL STROUT = $AB1E
220  -.GL CURSORHOME = $E566
230  -.GL GET = $FFE4
```

```
240  -.GL BASIN = $FFCF
250  -.GL BASOUT = $FFD2
260  -.GL RESET = $FCE2 ; SOFTWARE-RESET
270  -;
280  -START     JSR $E544       ; = PRINT CHR$(147)
290  -          LDA #0          ; TASTATURPUFFER
300  -          STA 198         ; LOESCHEN
310  -          STA MPT
320  -; ↑ SETZT AKTUELLEN MENUEPUNKT AUF 0
330  -HSCHLEIFE JSR CURSORHOME
340  -; ↑ HSCHLEIFE = HAUPTSCHLEIFE
350  -          LDA #0
360  -          TAX
370  -SCHLEIFE1 STA RVSTAB,X
380  -          INX
390  -          CPX #4
400  -          BNE SCHLEIFE1
410  -          LDX MPT
420  -          LDA #18         ; 18 = REVERS EIN
430  -          STA RVSTAB,X
440  -          LDX #0
450  -; ↑ SCHLEIFENZAEHLER INITIALISIEREN
460  -SCHLEIFE2 STX XSAVE       ; X RETTEN
470  -          LDA RVSTAB,X
480  -          JSR BASOUT
490  -          LDA TEXTLO,X    ; ERKLAERUNG
500  -          LDY TEXTHI,X    ; ZUM MENUEPUNKT
510  -          JSR STROUT      ; AUSGEBEN
520  -          LDX XSAVE       ; X WIEDER HOLEN
530  -          INX
540  -          CPX #4
550  -          BNE SCHLEIFE2
560  -;
570  -;
580  -; HIER IST DAS MENUE BEREITS AUF
590  -; DEN BILDSCHIRM AUSGEGEBEN WORDEN.
600  -;
610  -TASTE     JSR GET         ; TASTATURABFRAGE
620  -          BEQ TASTE       ; WARTEN AUF TASTENDRUCK
630  -          LDX #0
640  -SCHLEIFE3 CMP TASTEN,X
650  -          BEQ WEITER1
660  -          INX
670  -          CPX #16
680  -          BNE SCHLEIFE3
690  -          JMP TASTE
700  -WEITER1   TXA
710  -          LSR             ; DIVIDIERT AKKU-
720  -          LSR             ; MULATOR DURCH 4
730  -          TAX
740  -          LDA SP1LO,X
750  -          STA SPRUNG
760  -          LDA SP1HI,X
770  -          STA SPRUNG+1
780  -;
790  -.EQ RUECKSPRUNG = HSCHLEIFE-1
800  -; ↑ LEGT RUECKSPRUNGADRESSE DES
810  -;   UNTERPROGRAMMS FEST.
820  -;
830  -          LDA #>(RUECKSPRUNG)
840  -          PHA
850  -          LDA #<(RUECKSPRUNG)
860  -          PHA
870  -          JMP (SPRUNG)
```

```
880  -;
890  -;
900  -HOME       LDX #0
910  -           STX MPT
920  -ENDE       RTS              ; ENDE DES UNTERPRG
930  -;
940  -DOWN       LDX MPT          ; MENUEPUNKT
950  -           INX              ; UM 1 ERHOEHEN
960  -           CPX #4           ; GROESSER ALS 3?
970  -           BEQ HOME         ; DANN =0
980  -           STX MPT          ; SONST UEBERNEHMEN
990  -           RTS              ; ZUR HAUPTSCHLEIFE
1000 -;
1010 -UP         LDX MPT          ; MENUEPUNKT
1020 -           DEX              ; DEKREMENTIEREN
1030 -           BPL ENDUP        ; > 0?
1040 -           LDX #3           ; NEIN, DANN =3
1050 -ENDUP      STX MPT          ; UND UEBERNEHMEN
1060 -           RTS              ; ZUR HAUPTSCHLEIFE
1070 -;
1080 -;
1090 -EXEC       PLA              ; STAPELMANIPULATION
1100 -           PLA
1110 -           LDX MPT
1120 -           LDA SP2LO,X
1130 -           STA SPRUNG
1140 -           LDA SP2HI,X
1150 -           STA SPRUNG+1
1160 -           JMP (SPRUNG)
1170 -;
1180 -;
1190 -;
1200 -ZAHLWORT   LDA #<(TZAHL)    ; AUFFORDERUNG
1210 -           LDY #>(TZAHL)    ; ZUR EINGABE
1220 -           JSR STROUT       ; AUSGEBEN
1230 -           JSR BASIN        ; HOLT ZEICHEN
1240 -           SEC              ; IN BINAERZAHL
1250 -           SBC #"0"         ; UMWANDELN
1260 -           TAX              ; INS X-REGISTER
1270 -;
1280 -; JETZT STEHT IM X-REGISTER
1290 -; DIE EINGEGEBENE ZAHL
1300 -;
1310 -           CMP #10          ; > 10?
1320 -           BCC ZAHLWORT1    ; NEIN=> WEITER
1330 -           JMP ZAHLWORT     ; NEUEINGABE
1340 -;
1350 -ZAHLWORT1  STX XSAVE        ; X RETTEN
1360 -           LDA #<(TWORT)    ; TEXT >>IN WORTEN<<
1370 -           LDY #>(TWORT)    ; NACH EINGABE
1380 -           JSR STROUT       ; AUSGEBEN
1390 -           LDX XSAVE        ; X WIEDER HOLEN
1400 -           LDA ZWLO,X       ; ADRESSE DES
1410 -           LDY ZWHI,X       ; ZAHLWORTES HOLEN
1420 -           JSR STROUT       ; UND Z.WORT DRUCKEN
1430 -;
1440 -WAIT       JSR GET          ; WARTET AUF
1450 -           BEQ WAIT         ; TASTENDRUCK
1460 -           JMP START        ; ZUM HAUPTMENUE.
1470 -;
1480 -;
1490 -;
1500 -FARBE      LDA #<(TFARBE)
1510 -           LDY #>(TFARBE)
```

```
1520 -          JSR STROUT
1530 -          LDX #0
1540 -FARBE1    JSR BASIN       ; HOLT EINGABE
1550 -          CMP #" "        ; SPACE ?
1560 -          BEQ FARBE1      ; JA=>UEBERLESEN
1570 -          CMP #13         ; ENDE DER EINGABE?
1580 -          BEQ FARBE2      ; JA, DANN WEITER
1590 -          STA FARBWORT,X  ; EINGABE SPEICHERN
1600 -          INX             ; ZAEHLER ERHOEHEN
1610 -          JMP FARBE1      ; ZUR SCHLEIFE
1620 -FARBE2    STX 2           ; LAENGE MERKEN
1630 -          LDX #0
1640 -          TXA
1650 -FARBE3    ROL
1660 -          EOR FARBWORT,X
1670 -          INX
1680 -          CPX 2           ; SCHON FERTIG?
1690 -          BNE FARBE3      ; NEIN,ZUR SCHLEIFE
1700 -          CLC             ; LAENGE
1710 -          ADC 2           ; ADDIEREN
1720 -;
1730 -; HIER STEHT IM AKKU DIE PRUEFSUMME
1740 -;
1750 -          LDX #0
1760 -FARBE4    CMP PRUEFSUMMEN,X
1770 -          BEQ FARBE5      ; GEFUNDEN
1780 -          INX
1790 -          CPX #16
1800 -          BNE FARBE4
1810 -          JMP FARBE       ; NEUE EINGABE
1820 -FARBE5    STX 53280       ; BILDSCHIRM-
1830 -          STX 53281       ; FARBE SETZEN
1840 -          JMP START       ; ZUM MENUE
1850 -;
10000-;
10010-; TABELLEN
10020-; ========
10030-;
10040-; TEXTE:
10050-;
10060-PUNKT0     .TX "ZAHL IN ZAHLWORT UMWANDELN"
10070-.BY 13,13,0
10080-;
10090-PUNKT1     .TX "BILDSCHIRMFARBE"
10100-.BY 13,13,0
10110-;
10120-PUNKT2     .TX "RESET AUSLOESEN"
10130-.BY 13,13,0
10140-;
10150-PUNKT3     .TX "PROGRAMMENDE UEBER RTS"
10160-.BY 13,13,13
10170-.TX "BITTE AUSWAEHLEN !"
10180-.BY 0
10190-;
10200-;
10210-TASTEN     .BY 133,13,"←","="; 133=F1,13=RETURN
10220-.BY 19,"0","@",0 ; 19=HOME,0=DUMMY
10230-.BY 17,"D",135,"+" ; 17=CRSR DOWN,135=F5
10240-.BY 145,"U",134,"-" ; 145=CRSR UP,134=F3
10250-;
10260-;
10270-TZAHL      .BY 147          ; CLEAR HOME
10280-.TX "ZAHL (0-9) ? "
10290-.BY 0
```

```
10300-;
10310-TWORT      .TX " IN WORTEN : "
10320-.BY Ø
10330-;
10340-;
10350-; ZAHLWOERTER (Ø-9)
10360-;
10370-;
10380-NULL       .TX "NULL"
10390-.BY Ø
10400-;
10410-EINS       .TX "EINS"
10420-.BY Ø
10430-;
10440-ZWEI       .TX "ZWEI"
10450-.BY Ø
10460-;
10470-DREI       .TX "DREI"
10480-.BY Ø
10490-;
10500-VIER       .TX "VIER"
10510-.BY Ø
10520-;
10530-FUENF      .TX "FUENF"
10540-.BY Ø
10550-;
10560-SECHS      .TX "SECHS"
10570-.BY Ø
10580-;
10590-SIEBEN     .TX "SIEBEN"
10600-.BY Ø
10610-;
10620-ACHT       .TX "ACHT"
10630-.BY Ø
10640-;
10650-NEUN       .TX "NEUN"
10660-.BY Ø
10670-;
10680-;
10690-TFARBE     .BY 147         ; CLEAR HOME
10700-.TX "WELCHE FARBE ? "
10710-.BY Ø
10720-;
10730-;
10740-RVSTAB     .BY Ø,Ø,Ø,Ø     ; 4 BYTES RESERVIEREN
10750-;
10760-;
10770-; ZAHLEN:
10780-;
10790-; ADRESSEN DER TEXTE, DIE DIE
10800-; MENUEPUNKTE BESCHREIBEN
10810-;
10820-TEXTLO     .BY <(PUNKTØ),<(PUNKT1)
10830-.BY <(PUNKT2),<(PUNKT3)
10840-;
10850-TEXTHI     .BY >(PUNKTØ),>(PUNKT1)
10860-.BY >(PUNKT2),>(PUNKT3)
10870-;
10880-;
10890-; ADRESSEN DER ZAHLWOERTER
10900-;
10910-ZWLO       .BY <(NULL),<(EINS),<(ZWEI),<(DREI)
10920-.BY <(VIER),<(FUENF),<(SECHS),<(SIEBEN)
10930-.BY <(ACHT),<(NEUN)
```

```
10940-;
10950-ZWHI       .BY >(NULL),>(EINS),>(ZWEI),>(DREI)
10960-.BY >(VIER),>(FUENF),>(SECHS),>(SIEBEN)
10970-.BY >(ACHT),>(NEUN)
10980-;
10990-;
11000-; ADRESSEN DER UNTERROUTINEN
11010-; FUER DIE MENUESTEUERUNG
11020-;
11030-SP1LO      .BY <(EXEC),<(HOME),<(DOWN),<(UP)
11040-;
11050-SP1HI      .BY >(EXEC),>(HOME),>(DOWN),>(UP)
11060-;
11070-;
11080-; ADRESSEN DER EINZELNEN
11090-; MENUEPUNKTE
11100-;
11110-SP2LO      .BY <(ZAHLWORT),<(FARBE)
11120-.BY <(RESET),<(ENDE)  ; BEI ENDE STEHT
11130-SP2HI      .BY >(ZAHLWORT),>(FARBE)
11140-.BY >(RESET),>(ENDE)  ; EIN RTS-BEFEHL
11150-;
11160-; PRUEFSUMMEN DER FARB-WOERTER
11170-;
11180-PRUEFSUMMEN .BY 41,158,137,212,159,101
11190-.BY 3,2,33,69,201,116,113,121,127,114
11200-;
11210-;
11220-; ZWISCHENSPEICHER
11230-;
11240-MPT        .BY 0          ; 1 BYTE RESERVIEREN
11250-XSAVE      .BY 0
11260-SPRUNG     .WO 0          ; 2 BYTES FREIHALTEN
11270-FARBWORT   .BY 0
11280-; ↑ AB 'FARBWORT' WIRD DIE EINGABE
11290-;   DER FARB-BEZEICHNUNG ABGELEGT.
```

Zuerst soll die Bedienung des Programms erläutert werden. Gestartet wird der Objektcode »LISTING 10.11 (O)« über SYS49152, worauf man sich in folgendem Menü befindet:

ZAHL IN ZAHLWORT WANDELN	(0)
BILDSCHIRMFARBE	(1)
RESET AUSLOESEN	(2)
PROGRAMMENDE UEBER RTS	(3)
BITTE AUSWAEHLEN!	

Die Zahlen in Klammern sehen Sie am Bildschirm nicht, diese geben nur die interne Numerierung der Menüpunkte an.

Der jeweils angewählte Menüpunkt (unmittelbar nach dem Start: 0) wird im Gegensatz zu den anderen revers hervorgehoben. Der markierte Menüpunkt kommt durch Drücken von <F1>, <RETURN>, <←> oder <=> zur Ausführung. Wollen Sie einen anderen Menüpunkt wählen, drücken Sie einfach <CRSR DOWN>, <D>, <F5> oder <+>, um den invertierten Bereich nach unten zu verschieben. Weiter nach oben gelangen Sie über <CRSR UP>, <U>, <F3> oder <->.

Wenn Sie den obersten oder untersten Bereich »nach außen« überschreiten wollen, landen Sie jeweils an der entgegengesetzten Menü-Grenze; dies kennen Sie bereits aus den Menüs, die in Kapitel 7 vorgestellt wurden.
Sicher würden Sie Ihre Maschinenprogramme auch gerne mit einem solch komfortablen Menü aufwerten. Wenn Sie die Beschreibung des Quelltextes gut durchlesen, wird dies keine Schwierigkeiten bereiten.
Nun zu den einzelnen Menüpunkten.
»2« (Reset auslösen) springt in die Reset-Routine ab $FCE2. »3« (Programmende über RTS) bewirkt einen Rücksprung ins Basic. Wenn Sie aber »TABELLEN-BEISPIEL« vom Hypra-Ass aus gestartet haben, finden Sie sich möglicherweise im AUTONUMBER-Modus wieder; dies ist weder ein Fehler von »TABELLEN-BEISPIEL« noch von Hypra-Ass, sondern liegt daran, daß beide Programme eine bestimmte Adresse verwenden, deren unabsichtlich veränderten Inhalt der Hypra-Ass dann als Aufforderung zur automatischen Zeilennumerierung wertet. Am besten starten Sie »TABELLEN-BEISPIEL« nur vom normalen Basic aus.
Punkt »0« bittet Sie um Eingabe einer Zahl von 0 bis 9 und gibt zur eingegebenen Zahl das Zahlwort aus. Beispiel: Eingabe »0«, Ausgabe »NULL«. Danach müssen Sie eine Taste drücken, um ins Hauptmenü zu kommen.
Punkt »1« bietet die Möglichkeit, die Hintergrundfarbe besonders elegant einzustellen. Sie geben einfach die Farbe als Wort ein, zum Beispiel SCHWARZ. Folgende Eingaben sind vorgesehen: SCHWARZ, WEISS, ROT, TUERKIS, VIOLETT, GRUEN, BLAU, GELB, ORANGE, BRAUN, HELLROT, GRAU 1, GRAU 2, HELLGRUEN, HELLBLAU, GRAU 3. Aufgrund der Überschneidung von Prüfsummen zeigen jedoch auch andere Eingaben Wirkung (wenn diese auch nicht unbedingt orthographischen Vereinbarungen entsprechen), zum Beispiel: SCH, HYPRA ASS, PRINT, COMPUTER-GRAPHIK, TAGESSCHAU.
Nun wollen wir uns mit dem Quelltext befassen.
Ab Zeile 10000 finden Sie die Tabellen. Und weil unser Programm ein Beispiel für die Verwendung von Tabellen sein soll, sind es deren recht viele. Die wichtigsten davon sind jedoch analog der internen Numerierung der Menüpunkte aufgebaut, da sie Daten für die Menüsteuerung beinhalten. Diese Tabellen sind auch mit 0–3 numeriert und grafisch in Bild 10.5 dargestellt.
Sehen wir uns wieder den Quelltext an, beginnend mit der ersten Zeile.
Auf die Symboldefinitionen (210–260) folgt die Initialisierung der Hauptschleife (280–310). Diese Initialisierung löscht den Bildschirm (280) und Tastaturpuffer (290/300). Außerdem wird der aktuelle (= derzeit invers dargestellte) Menüpunkt, welcher immer in der Adresse MPT enthalten ist, auf 0 gesetzt (310). Zeile 310 ist also dafür verantwortlich, daß nach dem Start über SYS49152 das Inversfeld ganz oben auf Punkt 0 steht.
Die Texte, die der Beschreibung der Menüpunkte dienen, werden in der Hauptschleife HSCHLEIFE (350–550) ausgegeben. Mit dieser wollen wir uns nun eingehend auseinandersetzen. Zunächst wird die Tabelle RVSTAB gelöscht (350–400). Diese Tabelle enthält die Information, ob der erläuternde Text zu einem Menüpunkt invers ausgegeben wird. Wenn nein, so enthält das entsprechende Byte eine »0«, andernfalls eine »18« (= RVS-ON-Code für Betriebssystem). Das entsprechende Byte aus RVSTAB braucht nur vor dem Menüpunkt-Text ausgege-

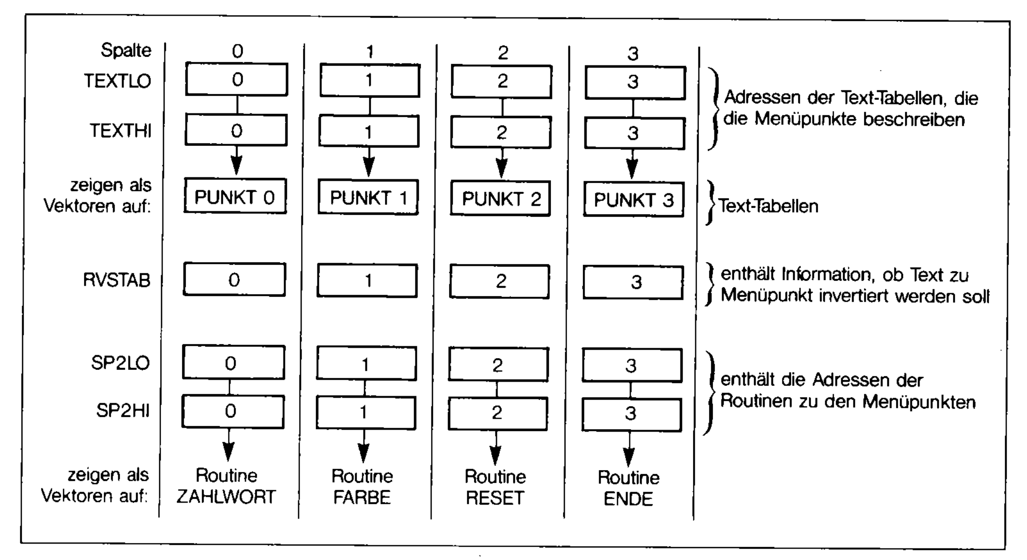


Bild 10.5: So verwendet man Tabellen zur Realisierung eines Menüs

ben werden (470–480). Die Zeilen 410–430 sorgen dafür, daß das Byte in RVSTAB, welches sich auf den aktuellen Menüpunkt bezieht, den RVS-ON-Code enthält. In der Hauptschleife muß das X-Register in XSAVE gesichert werden, weil die Routine STROUT den Inhalt des X-Registers ändert.

Mit TASTE (610) beginnt dann die Tastaturabfrage im Menü. Die Routine GET holt ein Zeichen von der Tastatur als ASCII-Code in den Akku; wurde keine Taste gedrückt, erhält der Akku den Code 0. In diesem Fall wartet 620 auf eine neue Eingabe. Beachten Sie bitte, daß der Akku nach der Zeile 620 nie (!) den Wert 0 haben kann (dies erweist sich bald als nützlich).

Wurde nun eine Taste gedrückt, sucht SCHLEIFE (630–680) in der Tabelle TASTEN, die im Quelltext ab Zeile 10210 steht, nach dem eingegebenen Zeichen; wird es nicht gefunden, erfolgt in Zeile 690 der Sprung zur neuen Eingabe. Die Tabelle TASTEN enthält alle vorgesehenen Tastendrücke zur Menüsteuerung, die in 4er-Blöcken angeordnet sind (Bild 10.5).

Nach der Suchschleife steht im X-Register die Position der gedrückten Taste innerhalb der Tabelle TASTEN (zum Beispiel 0 = <F1> gedrückt, 4 = <HOME> gedrückt). Diese Position wird – ohne Berücksichtigung des Divisionsrestes – durch 4 dividiert, um festzuhalten, von welchem Tastenblock eine Taste gedrückt wurde. Dadurch ist eindeutig bestimmt, welche Befehlsgruppe aufgerufen werden muß.

Steht nach 730 im X-Register 0, wurde eine der ersten vier in TASTEN enthaltenen Tasten gedrückt, die allesamt die Ausführung des aktuellen Menüpunktes veranlassen (Zeile 10210 und Bild 10.5). Ist X=1 so wurde eine Taste aus Zeile 10220 gedrückt. In 10220 steht als letztes Byte eine Null; diese dient, da für die Funktion »Inversfeld in HOME-Position« nur drei Tastendrücke vorgesehen wurden, zum Auffüllen auf vier Tasten. 0 kann hier bedenkenlos als Dummy

(Füllbyte ohne wirkliche Bedeutung) stehen, da der Akku aufgrund von 620 nie den Wert 0 an dieser Stelle haben wird. Beinhaltet X nach der Division durch 4 den Wert 2, wird das Inversfeld nach unten bewegt, ist X=3, dann nach oben. Diesen Zusammenhang veranschaulicht Bild 10.6.

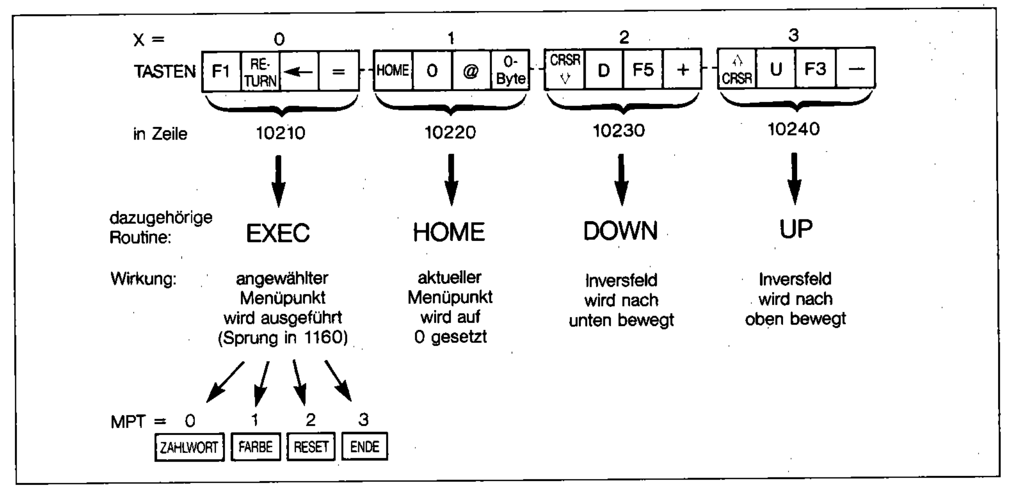


Bild 10.6: Die Tastaturabfrage aus Listing 10.11

An den Zeilen 740–870 sehen wir nun die Verwendung einer Sprungtabelle. Unsere Sprungtabelle ist SP1LO/SP1HI; SP1LO beinhaltet die Low-, SP1HI die High-Bytes der anzuspringenden Routinen. In den Vektor SPRUNG wird einfach die Zieladresse geschrieben (740–770). Die Zuweisungszeile 790 errechnet die Rücksprungadresse des aufzurufenden Unterprogramms. Bei einem RTS soll nämlich zur HSCHLEIFE zurückgesprungen werden. Diese Rücksprungadresse RUECKSPRUNG wird auf den Stapel gelegt (830–860), zuletzt erfolgt der indirekte Sprung (870). Die über die soeben beschriebene Simulation von JSR(ind) angesprungenen Routinen finden Sie ab Zeile 900. Es wird einfach der aktuelle Menüpunkt MPT entsprechend dem Tastendruck geändert, dann wird zur HSCHLEIFE gesprungen, die auch die Tabelle RVSTAB entsprechend anpaßt.
EXEC (1090) holt die Rücksprungadressen vom Stapel (1090–1100), da diese Routine nicht als Unterprogramm zu behandeln ist. Zeile 1110 holt den angeforderten Menüpunkt ins X-Register. Dann wird aus SP2LO/SP2HI die Adresse der zum Menüpunkt gehörenden Routine geholt und diese über einen gewöhnlichen indirekten Sprung aufgerufen (1160). Als Routine zu »2« wird einfach die Reset-Routine des Betriebssystems angesprungen, für »3« eignet sich jeder RTS-Befehl, also auch der bei ENDE (920).
ZAHLWORT, die Routine zu 0, holt eine Zahl als ASCII-Code (1230) und wandelt sie in einen numerischen Wert um (1240/1250), indem sie den ASCII-Code von »0« abzieht. Das Ergebnis landet im X-Register (1260). Ob auch eine Zahl eingegeben wurde, prüfen die Zeilen 1310-1330. Bei ZAHLWORT (1350) wird das Resultat der Subtraktion in XSAVE gesichert, der Text »IN WORTEN« ausgegeben und das X-Register wieder geholt.

Die Tabelle ZWLO/ZWHI enthält die Adressen, ab denen die Texte der Zahlwörter als ASCII-Code stehen. Aus ZWLO/ZWHI wird dann diese Adresse geholt (1400/1410) und der dort stehende Text ausgegeben (1420). Danach erwartet das Programm noch einen Tastendruck (1440–1450), bevor ins Hauptmenü verzweigt wird (1460).
Als letzte Routine ist FARBE (1500–1850) zu besprechen; hierzu ist jedoch aufgrund der Ähnlichkeit zu Listing 10.10 nicht viel zu erläutern. Bei 1820 steht im X-Register der Code der eingegebenen Farbe (= Position der Prüfsumme innerhalb der Tabelle PRUEFSUMMEN). Dieser muß nur noch in die entsprechenden VIC-Register geschrieben werden (1820/1830).
Ab Zeile 10000 stehen dann die bereits erwähnten Tabellen. Wenn Sie die Tabellen angesehen haben, sollten Sie durchaus noch einmal den Quelltext bis 10000 betrachten und die hier endende Programmbeschreibung lesen. Denn wenn Sie das Programm »TABELLEN-BEISPIEL« ganz verstanden haben, sind Sie in der Assemblerprogrammierung einen großen Schritt weitergekommen!

10.7 Die Nutzung der Zeropage

In jedem Assembler-Lehrbuch werden die Vorteile der Zeropage-Adressierung gepriesen. Speicherplatzersparnis und hohe Verarbeitungsgeschwindigkeit sind nicht die einzigen Vorzüge; die indirekt-indizierte Adressierung kann nur auf Zeropage-Adressen zugreifen, nicht auf absolute 16-Bit-Adressen. Damit wird der Leser aber schon alleine gelassen. Er erfährt nicht, welche Adressen in der Zeropage für die Praxis geeignet sind. Das wird nun nachgeholt.
Fast die ganze Zeropage wird durch Basic-Interpreter und Betriebssystem belegt. Deshalb führen bestimmte Werte in Zeropage-Adressen oft zu Absturz oder sonstigem Fehlverhalten des Computers. Wie dies im einzelnen aussieht, läßt sich mit Hilfe eines umfassenden Nachschlagewerkes bis ins letzte Detail klären: »C64 für Insider« dokumentiert in ROM-Listing und Text alle ROM-Routinen und die davon angesprochenen Adressen, enthält eine komplette Speicherbelegung (»Memory Map«) und erklärt an gegebenen Stellen auch die Verwendbarkeit von Speicherplätzen für eigene Programme; die allgemeine Erklärung der internen Zusammenhänge, aller wichtigen Programmiertechniken und Begriffe enthält ebenfalls zahlreiche Hinweise zu diesem Thema.
Das beste Hilfsmittel zur Entscheidung über den Einsatz bestimmter Adressen ist jedoch eine »Cross-Referenz« über den gesamten Speicher; diese gibt klipp und klar Auskunft, an welchen Adressen in welcher Weise Zugriffe auf bestimmte Adressen erfolgen. In »C64 für Insider« finden Sie die einzige Cross-Referenz über den gesamten Speicher des C64.
Ich möchte Ihnen nun an dieser Stelle zeigen, welche Adressen Sie als (Zwischen-)Speicher ohne Schwierigkeiten verwenden können, und was Sie bei Verwendung von Zeropage-Adressen beachten müssen. Dies wird Ihnen jegliche Orientierung im C64-Speicherdschungel, ähnlich einem Kompaß, erleichtern.

10.7.1 Problemlos verwendbar

Auf die Adressen $02 und $FB–$FE wird weder vom Basic-Interpreter noch vom Betriebssystem zugegriffen. Lediglich bei Initialisierung der Arbeitsspeicher (Reset) werden sie auf 0 gesetzt. Für die Praxis heißt das, daß Ihnen die genannten Adressen völlig zur Verfügung stehen.

10.7.2 In keiner Weise verwendbar

Von anderen Adressen hingegen müssen wir unsere Finger lassen. Diese haben entweder elementare Funktionen für Betriebssystem oder CPU, oder sie werden von beiden permanent geändert, so daß keine Datensicherheit besteht. Belassen Sie die Adressen $00 und $01 unverändert, da sie für die CPU wichtige Informationen über den Speicheraufbau enthalten und außerdem einige Bits nur durch externe Vorgänge geändert werden.
Von Bildschirmeditor und Tastaturabfrage werden die Adressen $C6–$F6 beeinflußt. Die Adressen $90–$C2 dienen der Ein-/Ausgabe-Steuerung mit Peripheriegeräten und der Verwaltung offener Files. Einzige Ausnahme: $A0–$A2, die Hilfsspeicher der internen Uhr. Somit sind diese elementaren Hilfsspeicher beim Ablauf von Programmen unter dem normalen Betriebssystem nicht verwendbar.
Wenn ein Maschinenprogramm in ein Basic-Programm als Unterroutine eingebaut ist, sind die Adressen $03–$56 sowie $73–$8B tabu.

10.7.3 Bedingt einsetzbar

Der Vektor $C3/$C4 wird durch <RUN/STOP RESTORE>, Reset oder LOAD beeinflußt. Ansonsten kann mit $C3/$C4 frei verfahren werden. Ganz Vorsichtige können diesen Vektor auf seinen Ausgangswert $FD30 setzen, sobald das Programm die Adressen $C3/$C4 nicht mehr für eigene Zwecke benötigt.

10.7.4 Bei Verzicht auf Kassettenbetrieb

Die folgenden Adressen können verwendet werden, wenn weder auf RS 232 noch auf Datasette zugegriffen wird: $9E/$9F, $A5–$A7, $A9–$AB, $B0–$B6, $F7–$FA.
Bei anderen Adressen, die sich auf den RS-232- oder Kassettenbetrieb beziehen, ist Vorsicht angebracht.

10.7.5 Geeignete Zwischenspeicher

Die Adressen $22–$2A und $57–$60 sind sogenannte »verschieden genutzte Arbeitsbereiche«. Sie werden vom Basic-Interpreter vor allem als Zwischenspeicher bei arithmetischen Operationen verwendet. Als solche Zwischenspeicher können auch wir sie einsetzen. Sobald allerdings bestimmte Interpreterroutinen aufgerufen werden, können die Inhalte dieser Adressen verlorengehen. Eine längerfristige Aufbewahrung von Daten in diesen Adressen ist zwar nicht möglich, andererseits können wir durch Schreibzugriffe auf diese Adressen das Betriebssystem oder den Basic-Interpreter nicht stören. Zu sagen wäre noch, daß die Adressen $57–$60 den wichtigen Routinen BLTUC und UMULT als Zwischenspeicher dienen.

10.7.6 Zeropage kopieren

Schließlich sei noch ein Trick verraten, der von einigen professionellen Programmen angewandt wird. Wir sichern die Zeropage-Inhalte in einem anderen Bereich, zum Beispiel von $6F00 an. Dann können wir viele Adressen in der Zeropage nutzen, solange wir keine Interpreter- oder Betriebssystemroutine aufrufen und auch den Systeminterrupt entweder gänzlich ausschalten (SEI) oder durch eine eigene Routine ersetzen.
Danach schreiben wir die Zeropage wieder von der Kopie, zum Beispiel $6F00, zurück und können wie gewöhnlich fortfahren. Die Adressen 0 und 1 kopieren wir nicht, weil diese nach wie vor für solche Zwecke nutzlos sind. Ebenso könnten wir nur einzelne Bereiche kopieren (zum Beispiel die Zeiger für Basic-Programme $16–$4A). Dann dürfen wir aber auch nur diesen Bereich verändern.
Da Sie auf diese Weise viel Speicherplatz in der Zeropage gewonnen haben, ist es sogar möglich, eine Tabelle aus Geschwindigkeitsgründen in die Zeropage zu verlegen. Damit steigt auch der Wert der indiziert-indirekten Adressierung erheblich.
Dennoch ist der Speicherplatz in der Zeropage begrenzt. Überlegen Sie sich also, auf welche Werte besonders schnell zuzugreifen ist, und schreiben Sie vorzugsweise diese in die Zeropage.

10.8 Schleifenprogrammierung

Die Programmierung effektiver Schleifen ist eine wichtige Basis für leistungsstarke Software.

10.8.1 Typ A: Schleifen mit maximal 256 Durchläufen

Da 256 verschiedene Zahlen mit einem 8-Bit-Prozessor darstellbar sind, verwendet man hier das X- oder Y-Register als Schleifenzähler. In Listing 10.12 sehen Sie die einfachste Form einer Schleife, die die Zeropage-Adressen $02-$FF nach $6F00 kopiert.

Listing 10.12: Inkrementierschleife

```
. 6000  A2 00     LDX #00
. 6002  B5 02     LDA 02,X
. 6004  9D 00 6F  STA 6F00,X
. 6007  E8        INX
. 6008  E0 FE     CPX #FE
. 600A  D0 F6     BNE 6002
```

Da der Schleifenzähler X in Listing 10.12 INKREMENTIERT wird, haben wir es mit einer INKREMENTIERSCHLEIFE zu tun. Nach dem Inkrementieren (»6007 INX«) wird durch »6008 CPX #FE« überprüft, ob die Schleife beendet werden kann. Eine eingehendere Beschreibung des Programmablaufs erübrigt sich. Für Schleifen des Typs A (maximal 256 Durchläufe) ist es aber meist vorteilhaft, eine DEKREMENTIERSCHLEIFE zu verwenden. Wie eine solche programmiert wird, sehen wir an Listing 10.13.

Listing 10.13: Dekrementierschleife

```
. 6000  A2 FE     LDX #$FE
. 6002  B5 01     LDA $01,X
. 6004  9D FF 6E  STA $6EFF,X
. 6007  CA        DEX
. 6008  D0 F8     BNE $6002
```

Listing 10.13 unterscheidet sich in der Wirkung nicht von Listing 10.12, obwohl man dies nicht unbedingt auf den ersten Blick erkennt. Deshalb soll dieses Listing näher besprochen werden. In Zeile 6000 erhält das X-Register den Inhalt $FE. Durch »6002 LDA 01,X« ist $02 die niedrigste Zeropage-Adresse, die kopiert wird. In Listing 10.12 ist 0 der niedrigste X-Wert. Die niedrigste Adresse aufgrund von »6002 LDA 02,X« ist also auch $02 (stimmt auffällig!). Warum $FF die höchste kopierte Zeropage-Adresse ist, können Sie nun selbst den Listings 12 und 13 entnehmen.
Listing 10.14 ist eine Dekrementierschleife, die die Kopie der Zeropage wieder von $6F00 nach $02 zurückholt.

Listing 10.14: Dekrementierschleife, kopiert von $6F00 nach $02

```
. 6000  A2 FE     LDX #$FE
. 6002  BD FF 6E  LDA $6EFF,X
. 6005  95 01     STA $01,X
. 6007  CA        DEX
. 6008  D0 F8     BNE $6002
```

Der Vorteil von Dekrementierschleifen vom Typ A ist, daß zum Erkennen der Abbruchbedingung (X=0) kein Vergleichsbefehl erforderlich ist, weil nach dem DEX-Befehl automatisch das Z-Flag gesetzt wird, wenn X Null wird. Das Entfallen des Vergleichsbefehls »CPX #« bringt eine Ersparnis von 2 Byte Speicherplatz sowie insgesamt 508 Taktzyklen Rechenzeit. Da jedoch bei 6002 eine Seitenüberschreitung (eine Seite entspricht 256 Byte) vorliegt, schrumpft der Zeitgewinn auf 254 Taktzyklen; dies ließe sich aber vermeiden, indem wir die Zeropage nach $6F01 kopieren, womit durch »6004 STA $6F00,X« keine Seitenüberschreitung auftreten würde.
Nun wollen wir noch einen Sonderfall behandeln: Dekrementierschleifen vom Typ A, bei denen der Ausgangswert für X kleiner als 129 ist. In Listing 10.15 sehen Sie eine Schleife, die den Bereich $16–$4A nach $6F00 kopiert, Listing 10.16 schreibt die Werte von $6F00 zurück nach $16.

Listing 10.15: Dekrementierschleife, kopiert von $16–$4A nach $6F00

```
. 6000  A2 34     LDX #$34
. 6002  B5 16     LDA $16,X
. 6004  9D 00 6F  STA $6F00,X
. 6007  CA        DEX
. 6008  10 F8     BPL $6002
```

Listing 10.16: Dekrementierschleife, kopiert von $6F00 nach $16–$4A

```
. 6000  A2 34     LDX #$34
. 6002  BD 00 6F  LDA $6F00,X
. 6005  95 16     STA $16,X
. 6007  CA        DEX
. 6008  10 F8     BPL $6002
```

Selbstverständlich hätten wir das Problem auch so lösen können wie in Listing 10.13. Wir wollen aber noch eine andere Konstruktion von Dekrementierschleifen kennenlernen, die in diesem Sonderfall möglich ist. Sehen wir uns also Listing 15 an.
Bei 6000 wird ins X-Register die Zahl geladen, die man zu $16 addieren muß, um $4A zu erhalten. Dadurch wird zunächst bei 6002 die Adresse $4A gelesen und nach $6F34 geschrieben. Bei 6007 wird dekrementiert. Neu ist der Verzweigungsbefehl; es wird das N-Flag überprüft. Ist X=$FF, wird das N-Flag gesetzt und »6008 BPL 6002« beendet die Schleife. Der niedrigste Wert von X, der innerhalb der Schleife vorkommt, ist demnach $00.
Der BPL-Befehl funktioniert nur, wenn der Ausgangswert von X < 129 ist. Andernfalls wäre nämlich nach dem Dekrementieren X > 127 und damit das N-Flag gesetzt. Dies aber hätte zur Folge, daß die Schleife nur ein einziges Mal durchlaufen würde.
Zur soeben behandelten Schleifenkonstruktion ist noch zu sagen, daß sie nicht effektiver als eine Lösung wie in Listing 10.13 ist. Allgemeine Gültigkeit hat aber folgende Regel für Schleifen vom Typ A:

Bei Schleifen mit 0–255 Durchläufen ist Dekrementieren effektiver als Inkrementieren. Bei 256 Durchläufen erweist sich Inkrementieren oft als besser.

An Listing 10.17 sehen wir ein Beispiel für den zweiten Satz der Regel. Listing 10.17 kopiert die 256 Speicherplätze des Stapels ($0100–$01FF) und den Stapelzeiger nach $6F00–$7000.

Listing 10.17: Stapel nach $6F00 kopieren

```
. 6000  A2 FF     LDX #$FF
. 6002  BD 00 01  LDA $0100,X
. 6005  9D 00 6F  STA $6F00,X
. 6008  CA        DEX
. 6009  D0 F7     BNE $6002
. 600B  AD 00 01  LDA $0100
. 600E  8D 00 6F  STA $6F00
. 6011  BA        TSX
. 6012  8E 00 70  STX $7000
```

Listing 10.18 schreibt den Stapel wieder zurück.

Listing 10.18: Stapel von $6F00 zurückkopieren

```
. 6000  A2 FF     LDX #$FF
. 6002  BD 00 6F  LDA $6F00,X
. 6005  9D 00 01  STA $0100,X
. 6008  CA        DEX
. 6009  D0 F7     BNE $6002
. 600B  AD 00 6F  LDA $6F00
. 600E  8D 00 01  STA $0100
. 6011  AE 00 70  LDX $7000
. 6014  9A        TXS
```

Die Dekrementierschleife (6000–600A) kopiert nur den Bereich $0101–$01FF; $0100 wird dabei nicht übertragen, dies geschieht erst in 600B–600F. Eine andere Möglichkeit wäre ein zeitraubender CPX #FF-Befehl nach »6008 DEX«. 6011–6013 sichert schließlich noch das SP-Register. Hier ist in der Tat eine Inkrementierschleife besser. Ändern wir Listing 10.17 also in Listing 10.17 V2:

Listing 10.17 V2: Verbessert durch Inkrementierschleife

```
            LDX #00
-LOOP       LDA 0100,X
-           STA 6F00,X
```

```
         INX           ; !!
-        BNE LOOP
-        TSX
-        STX 7000
```

Analog ergibt sich Listing 10.18 V2:

Listing 10.18 V2: Verbessert durch Inkrementierschleife

```
         LDX #00
-LOOP    LDA 6F00,X
-        STA 0100,X
         INX           ; !!
-        BNE LOOP
-        LDX 7000
-        TXS
```

In den »V2-Listings« habe ich diejenigen Befehle, die sich in der symbolischen Darstellung nicht von den Listings 10.17 und 10.18 in der ursprünglichen Fassung unterscheiden, mit »–« markiert; die Ausrufezeichen kennzeichnen die entscheidenden Inkrementierbefehle.

10.8.2 Typ B: Schleifen mit mehr als 256 Durchläufen

Während Schleifen des Typs A meist so schnell abgearbeitet werden, daß man es gar nicht wahrnimmt, dauern Typ-B-Schleifen oft eine oder mehrere Sekunden. Deswegen wollen wir hier versuchen, den Zeitbedarf von Typ-B-Schleifen zu verringern. Unsere erste Typ-B-Schleife (Listing 10.19) soll den Bereich von $3FD2 bis $475F invertieren (= EOR #FF-verknüpfen, aus jeder 1 wird eine 0 und umgekehrt).

Listing 10.19: Prototyp zur Invertierung von $3FD2–$475F

```
. 6000  A9 D2    LDA #$D2
. 6002  85 14    STA $14
. 6004  A9 3F    LDA #$3F
. 6006  85 15    STA $15
. 6008  A0 00    LDY #$00
. 600A  B1 14    LDA ($14),Y
. 600C  49 FF    EOR #$FF
. 600E  91 14    STA ($14),Y
. 6010  E6 14    INC $14
. 6012  D0 02    BNE $6016
. 6014  E6 15    INC $15
```

```
. 6016  A5 14   LDA $14
. 6018  C9 60   CMP #$60
. 601A  A5 15   LDA $15
. 601C  E9 47   SBC #$47
. 601E  90 EA   BCC $600A
```

Da hierfür ein 8-Bit-Indexregister nicht ausreicht, benötigen wir einen 16-Bit-Zähler, nämlich $14/$15. Dieser soll immer die Adresse beinhalten, die invertiert wird. In diesen Zähler schreibt die Initialisierung der Schleife den Startwert $3FD2 (s. $6000–$6007). Da es beim 6510 keine indirekte Adressierung für LDA/STA gibt, sondern nur die indirekt-indizierte oder indiziert-indirekte, müssen wir auf eine dieser Adressierungen ausweichen und dann den Index auf 0 setzen (»6008 LDY #00«).

Bei $600A beginnt die Schleife: Der Wert wird eingelesen mit $FF EOR-verknüpft und zurückgeschrieben. Nun wird der 16-Bit-Zähler $14/$15 erhöht (6010–6015). Dann wird überprüft, ob die nächste Adresse schon mit der ersten Adresse nach der Endadresse ($475F), also $4760, übereinstimmt (siehe 6016–601D). Dieser 16-Bit-Vergleich erfolgt durch eine Subtraktion nach einer Vergleichsoperation und ist sehr effektiv. Bei 601E wird schließlich die Schleife beendet, falls die Abbruchbedingung (C=1) erfüllt ist.

Listing 10.20 ist eine Dekrementierschleife, die sich in der Wirkung nicht von Listing 10.19 unterscheidet.

Listing 10.20: Dekrementierschleife zur Invertierung von $3FD2–$475F

```
. 6000  A9 5F   LDA #$5F
. 6002  85 14   STA $14
. 6004  A9 47   LDA #$47
. 6006  85 15   STA $15
. 6008  A0 00   LDY #$00
. 600A  B1 14   LDA ($14),Y
. 600C  49 FF   EOR #$FF
. 600E  91 14   STA ($14),Y
. 6010  A5 14   LDA $14
. 6012  D0 02   BNE $6016
. 6014  C6 15   DEC $15
. 6016  C6 14   DEC $14
. 6018  A5 14   LDA $14
. 601A  C9 D2   CMP #$D2
. 601C  A5 15   LDA $15
. 601E  E9 3F   SBC #$3F
. 6020  B0 E8   BCS $600A
```

Da das Dekrementieren einer 16-Bit-Adresse beim 6510 langsamer und speicherplatzaufwendiger ist als das Inkrementieren, ist Listing 10.20 weniger effektiv als Listing 10.19; in manchen Fällen benötigt man jedoch wegen der Programmlogik eine Dekrementierschleife, weshalb diese Technik hier ebenfalls vorgestellt wurde.
Grundsätzlich können Sie also an den Listings 10.19/10.20 sehen, wie man eine Typ-B-Schleife programmiert. Diese arbeitet jedoch nicht besonders schnell. Der Grund ist, daß der Bereich von $3FD2 bis $475F nicht restlos in ganze Seiten (256-Byte-Blöcke) aufgeteilt werden kann. Daher sollte man sich immer überlegen, ob sich die Schleifendurchlaufzahl nicht auf ganze 256-Byte-Blöcke »runden« läßt.
In unserem Fall würde dies heißen, daß mit einer schnelleren Schleife der exakt 8 x 256 Byte lange Bereich $3FD2-$47D1 invertiert wird, anstelle des ungeraden Bereichs $3FD2-$475F.
An einfacheren Zahlen wollen wir nun eine solche Schleife für ganze Seiten programmieren: Der 32 x 256 Byte umfassende Bereich von $2000 bis $3FFF (einschließlich) soll invertiert werden. Damit ließe sich eine Grafik, deren Bitmap in diesem Bereich liegt, invertieren. Die einfachste Lösung finden Sie in Listing 10.21.

Listing 10.21: Invertierung von $2000–$3FFF

```
. 6000  A9 00    LDA  #$00
. 6002  85 14    STA  $14
. 6004  A9 20    LDA  #$20
. 6006  85 15    STA  $15
. 6008  A0 00    LDY  #$00
. 600A  B1 14    LDA  ($14),Y
. 600C  49 FF    EOR  #$FF
. 600E  91 14    STA  ($14),Y
. 6010  C8       INY
. 6011  D0 F7    BNE  $600A
. 6013  E6 15    INC  $15
. 6015  A5 15    LDA  $15
. 6017  C9 40    CMP  #$40
. 6019  D0 EF    BNE  $600A
```

Zuerst wird die Anfangsadresse in $14/$15 abgelegt. Ins Y-Register kommt der Wert 0. Dann wird der Wert invertiert und das Y-Register, der Low-Zähler, erhöht. Ist der Wert noch nicht 0, wird die Schleife neu durchlaufen. Andernfalls wurde gerade eine Seite abgearbeitet. Der High-Zähler ($15) wird erhöht. Ist der Inhalt des High-Zählers = $40, wird die Schleife abgebrochen. Zu bemerken ist, daß während der Schleife die Adresse $14 unverändert 0 bleibt. Die Adresse, die jeweils invertiert wird, ergibt sich folgendermaßen:

```
(Y+Inhalt von $14) + 256*(Inhalt von $15)
```

Da wir auf die Adresse über das Prozessor-Register Y Einfluß nehmen können und die Adresse $14 nicht verändert werden muß, ist die Verarbeitungsgeschwindigkeit gegenüber der »Normalform« (Listing 10.20) gestiegen.
Das High-Byte müssen wir aber weiterhin in $15 belassen. Neu führen wir den High-Zähler X ein; im X-Register merken wir uns, wie viele Seiten invertiert werden. Diesen Wert verwenden wir als Dekrementierzähler. In unserem Fall werden $20 Seiten invertiert; weil $20 zufälligerweise auch das High-Byte der Anfangsadresse $2000 ist, wird dieser Wert in Listing 10.22 nur einmal (6005) in den Akku geladen und dann bei 6009 ins X-Register übertragen.

Listing 10.22: Invertierung von $2000–$3FFF mit High-Zähler X

```
. 6000  A9 00    LDA #$00
. 6002  85 14    STA $14
. 6004  A8       TAY
. 6005  A9 20    LDA #$20
. 6007  85 15    STA $15
. 6009  AA       TAX
. 600A  B1 14    LDA ($14),Y
. 600C  49 FF    EOR #$FF
. 600E  91 14    STA ($14),Y
. 6010  C8       INY
. 6011  D0 F7    BNE $600A
. 6013  E6 15    INC $15
. 6015  CA       DEX
. 6016  D0 F2    BNE $600A
```

Beachten Sie bitte, daß in Listing 10.22 die Befehle »6004 TAY« und »6009 TAX« nur bei den Werten dieses Beispiels verwendet werden können; in der Regel sind eigene Befehle »LDX #« oder »LDY #« erforderlich. Wollen wir zum Beispiel den Bereich $3FD2–$47D1 invertieren, so sieht die Initialisierung folgendermaßen aus:

```
LDA #D2    ; Low-Byte der ersten Adresse
STA 14
LDY #00    ; Index-Register
LDA #3F    ; High-Byte der ersten Adresse
STA 15
LDX #08    ; High-Zähler
... ...    ; Schleife wie ab 600A in Listing 10.22
```

Damit hätten wir eine Schleife, die den Bereich $3FD2–$475F (siehe Listings 10.19/10.20) invertiert und wesentlich schneller als die bisherigen Lösungen arbeitet. Da wir aber »aufge-

rundet« haben, wird zusätzlich der Bereich $4760–$47D1 invertiert, obwohl wir das gar nicht beabsichtigen. Es gibt nun drei Möglichkeiten, dieses Problem aus der Welt zu schaffen:

1. Wir verwenden die Schleife aus Listing 10.19, müssen aber eine deutlich höhere Arbeitsdauer hinnehmen.
2. Wir verwenden, wie gehabt, die Schleife aus Listing 10.22 mit der obigen Initialisierung. Dann invertiert eine Typ-A-Schleife den Restbereich $4760–$47D1 ein weiteres Mal. Damit wären – eine Besonderheit der EOR #FF-Verknüpfung – im Restbereich die alten Inhalte wiederhergestellt. Diese Lösung eignet sich aber fast nur bei dieser logischen Verknüpfung und hilft bei den meisten Typ-B-Schleifen nicht weiter.
3. Dies dürfte wohl die beste Lösung sein: Wir schreiben eine »gemischte« Schleife, die aus einer Typ-A- und einer Typ-B-Schleife besteht. Dieses Verfahren ist immer (!) möglich und wird von der BLTUC-Routine ($A3BF) des Basic-Interpreters angewandt; diese Verschieberoutine zerlegt sinnvollerweise den zu verschiebenden Bereich in einen Teil, der aus 256-Byte-Blöcken besteht und in einen Restbereich. Beide Bereiche werden dann getrennt verschoben.

Folgendermaßen sieht also die optimale Invertierroutine für den Bereich $3FD2–$475F aus:

- Der exakt 7 Seiten umfassende Bereich $3FD2–$46D1 wird mit einer Typ-B-Schleife wie in Listing 10.22 komplementiert.
- Der Restbereich $46D2–$475F wird mit einer Typ-A-Schleife wie in Listing 13 komplementiert.

Wir haben nun viele verschiedene Schleifenkonstruktionen in Theorie und Praxis beleuchtet. Was uns noch fehlt, sind formelhafte Rahmenprogramme, nach denen Sie die einzelnen Parameter (zum Beispiel für den X-Startwert in einer Dekrementierschleife vom Typ A) errechnen können. Als Zusammenfassung finden Sie deshalb in Form von Listing 10.23 ein Hypra-Ass-Assemblerlisting zu mehreren Schleifenkonstruktionen. Aus den Quelltext-Ausdrucken geht hervor, wie einzelne Parameter errechnet werden können.

Listing 10.23: Quelltext für mehrere Schleifenkonstruktionen

```
70   -.BA $C000
80   -.LI 1,3,0
90   -;
100  -; ******************************
110  -; *    QUELLTEXTE (HYPRA-ASS)   *
120  -; *    ====================     *
130  -; *                            *
140  -; * FUER VERSCHIEDENE SCHLEIFEN *
150  -; *                            *
160  -; * 28.08.85 BY FLORIAN MUELLER *
170  -; * SOWIE AENDERUNGEN 16.10.87  *
180  -; ******************************
190  -;
200  -;
```

```
210  -; QUELLTEXT ZU LISTING 10.12
220  -; ========================
230  -;
240  -.EQ ANFANGSADRESSE = $02
250  -.EQ ENDADRESSE = $FF
260  -.EQ ZIELBEREICH = $6F00
270  -;
280  -            LDX #0
290  -SCHLEIFE1 LDA ANFANGSADRESSE,X
300  -            STA ZIELBEREICH,X
310  -            INX
320  -            CPX #(ENDADRESSE+1-ANFANGSADRESSE)
330  -            BNE SCHLEIFE1
340  -;
350  -;
360  -; QUELLTEXT ZU LISTING 10.13
370  -; ========================
380  -;
390  -.EQ ANFANGSADRESSE = $02
400  -.EQ ENDADRESSE = $FF
410  -.EQ ZIELBEREICH = $6F00
420  -;
430  -            LDX #(ENDADRESSE+1-ANFANGSADRESSE)
440  -SCHLEIFE2 LDA ANFANGSADRESSE-1,X
450  -            STA ZIELBEREICH-1,X
460  -            DEX               ; DEKREMENTIERBEFEHL
470  -            BNE SCHLEIFE2
480  -;
490  -;
500  -; QUELLTEXT ZU LISTING 10.15
510  -; ========================
520  -;
530  -.EQ ANFANGSADRESSE = $16
540  -.EQ ENDADRESSE = $4A
550  -.EQ ZIELBEREICH = $6F00
560  -;
570  -            LDX #(ENDADRESSE-ANFANGSADRESSE)
580  -SCHLEIFE3 LDA ANFANGSADRESSE,X
590  -            STA ZIELBEREICH,X
600  -            DEX
610  -            BPL SCHLEIFE3   ; PRUEFT N-FLAG
620  -;
630  -;
640  -; QUELLTEXT ZU LISTING 10.19
650  -; ========================
660  -;
670  -.EQ ANFANGSADRESSE = $3FD2
680  -.EQ ENDADRESSE = $475F
690  -.EQ ZAEHLER = $14
700  -;
710  -            LDA #<(ANFANGSADRESSE)
720  -            STA ZAEHLER
730  -            LDA #>(ANFANGSADRESSE)
740  -            STA ZAEHLER+1
750  -            LDY #0
760  -SCHLEIFE4 LDA (ZAEHLER),Y
770  -            EOR #$FF
780  -            STA (ZAEHLER),Y
790  -            INC ZAEHLER
800  -            BNE WEITER
810  -            INC ZAEHLER+1
820  -WEITER    LDA ZAEHLER
830  -            CMP #<(ENDADRESSE+1)
840  -            LDA ZAEHLER+1
```

```
850  -            SBC #>(ENDADRESSE+1)
860  -            BCC SCHLEIFE4
870  -;
880  -;
890  -; QUELLTEXT ZU LISTING 10.21
900  -; ========================
910  -;
920  -.EQ ANFANGSADRESSE = $2000
930  -.EQ ENDADRESSE   = $3FFF
940  -.EQ ZAEHLER = $14
950  -;
960  -            LDA #<(ANFANGSADRESSE)
970  -            STA ZAEHLER
980  -            LDA #>(ANFANGSADRESSE)
990  -            STA ZAEHLER+1
1000 -            LDY #0
1010 -SCHLEIFE5 LDA (ZAEHLER),Y
1020 -            EOR #$FF
1030 -            STA (ZAEHLER),Y
1040 -            INY
1050 -            BNE SCHLEIFE5
1060 -            INC ZAEHLER+1
1070 -            LDA ZAEHLER+1
1080 -            CMP #>(ENDADRESSE+1)
1090 -            BNE SCHLEIFE5
1100 -;
1110 -;
1120 -; QUELLTEXT ZU EINER SCHLEIFE,
1130 -; DIE DEN BEREICH $3FD2-$47D1
1140 -; KOMPLEMENTIERT
1150 -;
1160 -.EQ ANFANGSADRESSE = $3FD2
1170 -.EQ ENDADRESSE = $47D1
1180 -.EQ ZAEHLER = $14
1190 -;
1200 -            LDA #<(ANFANGSADRESSE)
1210 -            STA ZAEHLER
1220 -            LDA #>(ANFANGSADRESSE)
1230 -            STA ZAEHLER+1
1240 -            LDX #>(ENDADRESSE+1-ANFANGSADRESSE)
1250 -            LDY #0
1260 -SCHLEIFE6 LDA (ZAEHLER),Y
1270 -            EOR #$FF
1280 -            STA (ZAEHLER),Y
1290 -            INY
1300 -            BNE SCHLEIFE6
1310 -            INC ZAEHLER+1
1320 -            DEX
1330 -            BNE SCHLEIFE6
1340 -;
1350 -; ENDE DES LISTINGS
```

10.9 Selbstmodifikation

Bevor wir uns mit dieser Programmiertechnik beschäftigen, die zwar nicht strukturiert, aber sehr trickreich ist, soll der Begriff geklärt werden. Unter Modifikation versteht man eine »Änderung, Anpassung«. Wenn Sie bei einem Spiel einen der vielen POKE-Befehle, die Sie in Kapitel 12 finden, eingeben, so wird dadurch das Spiel »modifiziert«. Die Änderung liegt dabei beispielsweise darin, daß man mehr Spielfiguren erhält oder das Spiel auch bei einem »Crash« beide Augen zudrückt.
Selbstmodifikation bedeutet folglich, daß sich ein Programm selbst verändert. Dies wäre der Fall, wenn im Spielprogramm eine Passage stünde, die den POKE ausführt.
Wenn Sie sich für die Selbstmodifikation von Basic-Programmen interessieren, so finden Sie in 2.5 (Tastaturpuffer-Anwendungen) genügend Informationen, Anregungen und Beispiele. Wir werden uns in diesem Abschnitt daher ausschließlich mit der Selbstmodifikation von Maschinenprogrammen befassen; dazu sei gleich gesagt, daß diese Programmiertechnik in Assembler nicht nur leistungsfähiger, sondern auch leichter zu realisieren ist als in Basic!
Als erstes Beispiel nehmen wir Listing 10.24.

Listing 10.24: Das erste selbstmodifizierende Maschinenprogramm

```
. 6000  A0 00     LDY #$00
. 6002  B9 00 20  LDA $2000,Y
. 6005  49 FF     EOR #$FF
. 6007  99 00 20  STA $2000,Y
. 600A  C8        INY
. 600B  D0 F5     BNE $6002
. 600D  EE 04 60  INC $6004
. 6010  EE 09 60  INC $6009
. 6013  AD 09 60  LDA $6009
. 6016  C9 40     CMP #$40
. 6018  D0 E8     BNE $6002
```

Es handelt sich um eine selbstmodifizierende Schleife, die den Bereich $2000–$3FFF komplementiert (mit EOR #FF verknüpft, also invertiert; in 10.8 war ein entsprechendes Programm ohne Selbstmodifikation bereits zu finden).
TRACEn Sie doch einmal durch Listing 10.24 mit dem SMON oder einem anderen Monitor und vergleichen Sie die disassemblierten Werte mit den ursprünglichen, wie sie in obigem Listing stehen. Sie werden schon nach einer minimalen Anzahl von TRACE-Schritten merken, daß die Befehle »6002 LDA 2000,Y« und »6007 STA 2000,Y« aufgrund der INC-Befehle immer auf andere Adressen zugreifen; bald heißt es »6002 LDA 2100,Y« und »6007 STA 2100,Y«, kurz darauf »6002 LDA 2200,Y« und »6007 STAI2200,Y« und so weiter.

Besagte INC-Befehle erhöhen also jeweils das High-Byte des Operanden. Ist dieses schon $40, wird die Schleife beendet. In Listing 10.25 sehen Sie, wie unsere Schleife aus Listing 24 aussieht, wenn sie fertig durchlaufen wurde.

Listing 10.25: So sieht Listing 10.24 nach einmaliger Ausführung aus

```
. 6000  A0 00     LDY #$00
. 6002  B9 00 40  LDA $4000,Y
. 6005  49 FF     EOR #$FF
. 6007  99 00 40  STA $4000,Y
. 600A  C8        INY
. 600B  D0 F5     BNE $6002
. 600D  EE 04 60  INC $6004
. 6010  EE 09 60  INC $6009
. 6013  AD 09 60  LDA $6009
. 6016  C9 40     CMP #$40
. 6018  D0 E8     BNE $6002
```

Ein weiterer Start bewirkt, daß das Programm sich früher oder später selbst invertieren will und infolge dieses selbstmörderischen Verhaltens abstürzt. Was nämlich unserem Listing 10.24 noch fehlt, ist eine Initialisierung, die jedesmal den Ausgangswert ($2000) in die LDA/STA-Befehle einsetzt. In Listing 10.26 sehen Sie eine solche Initialisierung (6000–600F).

Listing 10.26: Diesmal wird initialisiert

```
. 6000  A9 00     LDA #$00
. 6002  8D 13 60  STA $6013
. 6005  8D 18 60  STA $6018
. 6008  A9 20     LDA #$20
. 600A  8D 14 60  STA $6014
. 600D  8D 19 60  STA $6019
. 6010  A0 00     LDY #$00
. 6012  B9 FF FF  LDA $FFFF,Y
. 6015  49 FF     EOR #$FF
. 6017  99 FF FF  STA $FFFF,Y
. 601A  C8        INY
. 601B  D0 F5     BNE $6012
. 601D  EE 14 60  INC $6014
. 6020  EE 19 60  INC $6019
. 6023  AD 19 60  LDA $6019
. 6026  C9 40     CMP #$40
. 6028  D0 E8     BNE $6012
```

Die Adresse $FFFF (bei 6012 und 6017) ist dabei ein Dummy-Wert, d.h., er dient nur zum vorläufigen Ausfüllen von Speicherplätzen und hat keine programmtechnische Bedeutung (darf er auch gar nicht haben, denn nach jedem Durchlauf der Schleife steht dort ein anderer Wert!). Der Dummy-Wert wird sogar schon von der Initialisierung überschrieben; wir hätten also statt $FFFF auch $040C oder andere verwenden können. Wichtig ist nur, daß »LDA dummy,Y« 3 Byte belegt.
Ein besonderer Vorteil der Selbstmodifikation ist es, daß selbstmodifizierende Schleifen keine Zähler in der Zeropage benötigen, weil der Zähler praktisch im Programm selbst liegt. In puncto Geschwindigkeit sind selbstmodifizierende Schleifen den herkömmlichen aber oft unterlegen.
Ein weiterer Vorteil von ihnen ist aber, daß man außer den wenigen Zeropage-Speicherplätzen auch mit weniger Registern auskommen kann; beim spärlichen 6510-Registersatz von drei Stück (A, X und Y) ist dies ein äußerst wichtiger, ja oft sogar »lebenswichtiger« Aspekt für einzelne Unterroutinen!
Listing 10.27 invertiert beispielsweise den Bereich $3FD2–$475F (die Aufgabenstellung kennen Sie aus 10.8); X- und Y-Register sowie Zeropage bleiben dabei unverändert, lediglich der Akkumulator fungiert als Arbeitsregister.

Listing 10.27: Alles mit Akkumulator-Operationen

```
. 6000  A9 D2     LDA  #$D2
. 6002  8D 11 60  STA $6011
. 6005  8D 16 60  STA $6016
. 6008  A9 3F     LDA  #$3F
. 600A  8D 12 60  STA $6012
. 600D  8D 17 60  STA $6017
. 6010  AD 00 00  LDA $0000
. 6013  49 FF     EOR  #$FF
. 6015  8D 00 00  STA $0000
. 6018  EE 11 60  INC $6011
. 601B  EE 16 60  INC $6016
. 601E  D0 06     BNE $6026
. 6020  EE 12 60  INC $6012
. 6023  EE 17 60  INC $6017
. 6026  AD 11 60  LDA $6011
. 6029  C9 60     CMP  #$60
. 602B  AD 12 60  LDA $6012
. 602E  E9 47     SBC  #$47
. 6030  90 DE     BCC $6010
```

Listing 10.28 kopiert den Basic-Interpreter ins RAM an gleicher Adresse, wobei nur das X-Register verwendet wird (!!).

Listing 10.28: Ein x-beliebiges Register ist mehr als genug...

```
. 6000  A2 00     LDX #$00
. 6002  8E 11 60  STX $6011
. 6005  8E 14 60  STX $6014
. 6008  A2 A0     LDX #$A0
. 600A  8E 12 60  STX $6012
. 600D  8E 15 60  STX $6015
. 6010  AE 00 00  LDX $0000
. 6013  8E 00 00  STX $0000
. 6016  EE 11 60  INC $6011
. 6019  EE 14 60  INC $6014
. 601C  D0 F2     BNE $6010
. 601E  EE 12 60  INC $6012
. 6021  EE 15 60  INC $6015
. 6024  AE 12 60  LDX $6012
. 6027  E0 C0     CPX #$C0
. 6029  D0 E5     BNE $6010
```

Nun wollen wir sehen, wie man bei der Entwicklung selbstmodifizierender Programme unter Zuhilfenahme eines guten Assemblers wie des Hypra-Ass vorzugehen hat. Zunächst einmal müssen diejenigen Stellen, an denen Modifikationen vorgenommen werden, mit Label definiert werden. Von diesen Label aus können die Stellen im Speicher, die geändert werden sollen, leicht berechnet werden.

Befehlscode	= LABEL + 0 =	LABEL
Low-Operand	=	LABEL + 1
High-Operand	=	LABEL + 2

Bei 2-Byte-Befehlen wird der Parameter wie der Low-Operand eines 3-Byte-Befehls errechnet. Als Beispiel finden Sie in Form von Listing 10.29 einen Quelltext (Assembler: Hypra-Ass) für Listing 10.28.

Listing 10.29: Quelltext für Listing 10.28

```
80   -.BA $6000
90   -.LI 1,3,0
100  -;
110  -; HYPRA-ASS-QUELLTEXT ZU EINER
```

```
120  -; SELBSTMODIFIZIERENDEN SCHLEIFE
130  -; (ARBEITET WIE LISTING 10.28)
140  -;
150  -; 1985 BY FLORIAN MUELLER
160  -;
170  -;
180  -.GL START = $A000
190  -.GL ENDE  = $BFFF
200  -;
210  -            LDX #<(START)
220  -            STX MOD1+1
230  -            STX MOD2+1
240  -            LDX #>(START)
250  -            STX MOD1+2
260  -            STX MOD2+2
270  -MOD1        LDX $FFFF
280  -MOD2        STX $FFFF
290  -            INC MOD1+1
300  -            INC MOD2+1
310  -            BNE MOD1
320  -            INC MOD1+2
330  -            INC MOD2+2
340  -            LDX MOD1+2
350  -            CPX #>(ENDE+1)
360  -            BNE MOD1
```

Während in Listing 10.28 der Ausgangswert bei 6010 »LDX 0000« und bei 6013 »STX 0000« ist, wurde im Quelltext $FFFF verwendet (270/280), um den Assembler ausdrücklich darauf hinzuweisen, den Dummy-Wert als 16-Bit-Adresse abzulegen und nicht als Zeropage-Adresse, wodurch der Befehl nur 2 statt 3 Byte belegen würde.
Die Stellen, die modifiziert werden, wurden mit MOD1 und MOD2 definiert. MOD1 ist zugleich der Schleifenbeginn; die BNEs in 310 und 360 haben also nichts mit der Modifikation im Sinn, sondern orientieren sich an MOD1 als Schleifenbeginn.

10.9.1 Anwendung auf absolute Adressierung

Bei der Stapelmanipulation haben wir schon ein Verfahren kennengelernt, den Befehl JSR(indirekt), der im normalen 6510-Befehlssatz nicht existiert, zu simulieren. Folgendermaßen kann über Selbstmodifikation ein Unterprogramm ab ADRESSE aufgerufen werden:

```
                LDA #<ADRESSE          ; Low-Byte der Adresse in
                STA SPRUNGBEFEHL+1     ; Low-Operand
                LDA #>ADRESSE          ; High-Byte der Adresse in
                STA SPRUNGBEFEHL+2     ; High-Operand
SPRUNGBEFEHL    JSR $FFFF              ; $FFFF = Dummy
```

Genauso kann man mit dem JMP-Befehl verfahren. Sogar bei den Schiebe-, Dekrementier- und Inkrementierbefehlen, die wie JSR keine indirekte Adressierung haben, ist auf diese Weise eine Simulation der indirekten Adressierung möglich.

Wird eine Sprungtabelle per Selbstmodifikation verarbeitet, müssen die Sprungadressen in der Tabelle nicht (!) dekrementiert werden.

10.9.2 Anwendung auf Immediate-Befehle

Oft müssen Werte, die berechnet werden, auf dem Stapel oder im Speicher abgelegt und dann, wenn man sie braucht, wieder aufgenommen werden. Ein Beispiel hierfür ist der »Basic-Start-Generator« (64'er, Ausgabe 7/85, Seite 74). Bei Erwähnung dieses Programms taucht natürlich die Frage auf, ob es sich noch um ein selbstmodifizierendes Programm handelt oder ob der »Basic-Start-Generator« nicht eher zu den Programmgeneratoren zählt. Diese Frage ist voll berechtigt, weshalb wir kurz auf sie eingehen wollen.
Der »Basic-Start-Generator« ist eindeutig den Programmgeneratoren zuzuordnen, da der generierte Programmteil nie angesprungen wird und somit ein eigenständiges Programm darstellt. Das Programm modifiziert also nicht sich selbst, sondern vielmehr ein zweites Programm, welches dann vom Benutzer gespeichert werden kann.
Die Programmierung ist aber bei Programmgeneratoren nicht anders als bei selbstmodifizierenden Programmen. Auf den Unterschied Programmgeneration/Selbstmodifikation werden wir an späterer Stelle näher eingehen.
Zunächst wollen wir aber ein praktisches Beispiel für die Anwendung der Modifikation von Immediate-Befehlen behandeln. Oft steht man vor dem Problem, ein Register zu sichern und später wieder zu holen. Im Falle des Akkumulators sieht das auf konventionellem Wege so aus:

```
PHA       ; Akku sichern
...       ; weiteres Programm
PLA       ; Akku wieder holen
```

Beim X-Register, das ja nicht direkt auf den Stapel gelegt werden kann (»PHX« müßte der entsprechende Befehl lauten), wird's schon ungünstiger:

```
TXA       ; X-Register in Akku
PHA       ; Akku sichern
...       ; weiteres Programm
PLA       ; Akku wieder holen
TAX       ; Akku ins X-Register
```

Hier wird also zusätzlich der Akku beeinflußt. Wenn dies, wie in zahlreichen Fällen, strikt zu vermeiden ist, wird folgender Weg gewählt:

```
STX $02   ; $02 = Zwischenspeicher
...       ; weiteres Programm
LDX $02   ; X wieder holen
```

Für die Sicherung des X-Registers gibt es aber noch eine weitere Lösung, die den X-Wert im Programm ablegt und dadurch nicht den Stapel oder irgendeinen anderen Zwischenspeicher außerhalb des Programms in Anspruch nimmt:

```
          STX GETX+1  ; X direkt in Befehl schreiben
          ...         ; weiteres Programm
GETX      LDX #$00    ; $00 = Dummy-Wert
```

Obiges Beispiel läßt sich sehr leicht auf Akkumulator oder Y-Register umschreiben:

```
          STA GETA+1  ; A direkt in Befehl schreiben
          ...         ; weiteres Programm
GETA      LDA #$00    ; $00 = Dummy-Wert
```

Folgendermaßen kann nun das X-Register mit dem Akkumulator verglichen werden:

```
          STX VGL+1   ; in Vergleichsbefehl ablegen
          (....       ; evtl. weitere Befehle)
VGL       CMP #$00    ; $00 = Dummy
```

Ohne Selbstmodifikation wäre ein Zwischenspeicher erforderlich:

```
          STX $02     ; in Zwischenspeicher schreiben
VGL       CMP $02     ; Vergleich mit Zwischenspeicher
```

Als letztes Beispiel für die Anwendung auf Immediate-Befehle soll das Y-Register zum Akkumulator addiert werden:

```
          STY ADD+1   ; in Arithmetikbefehl ablegen
          (....       ; evtl. weitere Befehle)
          CLC         ; Carry vor Addition löschen
ADD       ADC #$FF    ; $FF = Dummy
```

Die Anwendungsmöglichkeiten sind hier wirklich unbegrenzt, und obendrein erleichtern sie in vielen Situationen die Programmierung erheblich.

10.9.3 Anwendung auf komplette Befehle

Bisher haben wir nur die Parameter einzelner Befehle modifiziert. Es ist selbstverständlich auch möglich, die Befehlscodes oder die kompletten Befehle zu modifizieren. Wenn nur der Befehlscode geändert wird (zum Beispiel ein »ORA #« in einen »EOR #«-Befehl), bleiben die Parameter sogar erhalten. Es könnte ferner ein impliziter Befehl (SEI, CLI, CLD, DEX, INX, ...) geändert werden, um beispielsweise zwischen In- und Dekrementieren umzuschalten oder das Setzen eines Prozessorflags durch das adäquate Löschen zu ersetzen. Außerdem könnte bei einem BRANCH-Befehl die Sprungbedingung (CS, CC, VS, VC, NE, EQ) geändert werden; aus BCS würde also mit Leichtigkeit BCC.

Mit diesem Wissen lösen wir noch das häufig auftretende Problem, wie die Ausführung eines Unterprogramms verhindert wird. Dazu werden wir drei unterschiedliche Lösungen (1–3) entwickeln.

1. Die Adresse FLAG wird auf 0 gesetzt, wenn das Unterprogramm ausgeführt werden soll, und auf einen anderen Wert, wenn es nicht ausgeführt werden soll.

```
          LDA #0              ; Flag für Ausführung
          STA FLAG            ; Flag setzen
          (.......            ; evtl. weiteres Programm)
          LDA FLAG            ; Flag testen
          BNE NEIN            ; Flag<>0, also nicht ausführen
          JSR UNTERPROGRAMM   ; Aufruf
NEIN      ...                 ; weiteres Programm
```

   Das Flag könnte auch am Beginn des Unterprogramms abgefragt und dann, wenn FLAG < >0 ist, das Unterprogramm unverzüglich verlassen werden.

2. Als ersten Befehl des Unterprogramms verwenden wir NOP:

```
UP        NOP         ; Beginn des Unterprogramms
          ...         ; Fortsetzung des Unterprogramms
```

   So wird die Ausführung des Unterprogramms gestattet:

```
          LDA #$EA  ; Opcode für NOP
          STA UP    ; an Anfang des Unterprogramms schreiben
```

   Und so wird sie verhindert:

```
          LDA #$60  ; Opcode für RTS
          STA UP    ; an Anfang des Unterprogramms schreiben
```

   Wer noch einen NOP-Befehl und damit 1 Byte sparen möchte, kann den NOP-Befehl entfallen lassen. Dann muß auch der Opcode $EA beim Erlauben des Unterprogramms in den Opcode des ersten Bytes im Unterprogramm geändert werden. Weil dies ziemlich mühselig ist, ziehe ich die Lösung mit NOP trotz des um 1 Byte erhöhten Speicherbedarfs vor.

3. Das beste Verfahren: Wir schalten den JSR-Befehl aus, indem wir ihn in einen BIT-Befehl abändern.

   Dazu definieren wir den Aufruf des Unterprogramms als Label:

```
AUFRUF    JSR UNTERPROGRAMM
```

   JSR ausschalten geschieht nun über:

```
          LDA #$2C  ; Opcode für BIT $xxxx
          STA AUFRUF
```

JSR wieder zulassen:

```
LDA #$20  ; Opcode für JSR $xxxx
STA AUFRUF
```

Der JSR-Opcode muß nicht unbedingt mit $2C überschrieben werden, auch $0C wäre möglich. $0C ist ein »undefinierter Opcode« für ein 3-Byte-NOP (also »no operation« unter Ignorieren der zwei darauffolgenden Bytes) und arbeitet mit allen mir bekannten Versionen des C 64 und C 128.

Im übrigen können mit dem soeben beschriebenen Verfahren auch andere Befehle ausgeschaltet werden, zum Beispiel JMP, LDA, STA und so weiter. Wenn aber der JSR-Opcode mit $2C (BIT-Opcode) überschrieben wird, ist darauf zu achten, daß bei der Ausführung des BIT-Befehls die Prozessorflags gesetzt werden. Sicherlich gibt es noch mehr Problemlösungen als 1–3, aber 3 dürfte wohl kaum zu übertreffen sein.

10.9.4 Anwendung auf mehrere Befehle

Selbstverständlich können ganze Befehlsfolgen, also größere Programmteile, gegeneinander ausgetauscht werden. Zu beachten ist nur, daß die Routinen, die gegeneinander auszutauschen sind, auch in dem Bereich, in den sie vom Programm aus geschrieben werden, lauffähig sind. Dies ist vor allem dann gegeben, wenn nur relative Adressierung bei Sprüngen innerhalb des Programms verwendet wird und dadurch die Routine im Speicher frei verschiebbar ist.
Bei längeren Routinen hätte dies auch den Vorteil, daß die »auf Reserve« abgelegten Routinen bei Bedarf aus einem »gepackten« Zustand ins ablauffähige Befehlsformat zu konvertieren wären.

10.9.5 Anwendung auf Tabellen

Dieser Anwendungsfall würde auch zum Abschnitt über »Tabellen« passen. Wir bleiben hier bei der Theorie, denn die Umsetzung in ein Programm wird in größerem Rahmen in 10.9.6 erfolgen.

- Im Zeichenprogramm Hi-Eddi liegt eine Tabelle, die die High-Bytes der Bit-Map-Anfangsadressen beinhaltet. Diese Tabelle wird von Hi-Eddi bei jedem Bildwechsel umgerechnet.
- Besonders flexible Programme erlauben Eingriffe des Anwenders in die Befehls- oder Text-Tabellen. So können Bildschirmmasken editiert oder Eingabemasken erstellt werden. Ein solches Programm braucht sich nach den Modifikationen nur selbst abzuspeichern. Weil hier unter Umständen ein erheblicher Teil des Programmschutzes in Gefahr gerät, werden oftmals lediglich die Tabellen abgespeichert.

- Ein Adventure-Generator modifiziert in der Regel auch nur die Tabellen eines fertigen Adventureprogramms, die eigentlichen Befehle, aus denen sich das Programm zusammensetzt, bleiben unverändert. In den Tabellen sind dann die einzelnen Spielsituationen und -informationen enthalten.
- Unter GEOS (siehe »C64 – Alles über GEOS 1.2«, S.342; »C64 – Alles über GEOS 1.3«) erlaubt das GEOS-Kernel, jederzeit den verwendeten Zeichensatz zu wechseln. Dabei liegt im Bereich $26–$2D eine Reihe von Hilfszeigern, die die Eigenschaften des aktuellen Zeichensatzes genau bezeichnen: Adresse der Zeichen-Bitmaps, Höhe der Zeichen in Pixel usw. Diese Werte werden nun bei Zeichensatzwechseln jeweils aus der Zeichensatz-Datei (»font file«) in die entsprechende Tabelle $26–$2D kopiert.

Gerade am letzten Beispiel sieht man, daß es bei Tabellen prinzipiell zwei verschiedene Arten sind, deren Unterscheidungskriterium uns nun bekannt ist:

- Konstante Tabellen enthalten während des gesamten Programmablaufs dieselben Werte. Dies sind also alle diejenigen Zahlen und Zeichen, die nicht zur Veränderung bestimmt sind.
- Arbeitstabellen werden je nach Bedarf modifiziert; wird aufgrund einer speziellen Programmsituation (Beispiel für GEOS: es findet kein Zeichensatzwechsel statt) keine Veränderung daran vorgenommen, so handelt es sich nur in der Theorie um Arbeitstabellen, in der Praxis jedoch um konstante Tabellen.

10.9.6 Das Beispielprogramm »Loader-Maker 64«

Wie aus dem Namen des Beispielprogramms schon zu entnehmen ist, handelt es sich um einen Programmgenerator. Da – wie gesagt – die Programmierung wie bei selbstmodifizierenden Programmen ist, habe ich bewußt einen Programmgenerator als Beispiel gewählt; schließlich ist dieses Thema auch sehr interessant und rückt gerade durch das Interesse an »künstlicher Intelligenz« immer mehr ins Blickfeld der Öffentlichkeit.
Unter dem Namen »LISTING 10.31« finden Sie auf Diskette den Objektcode (trotz Basic-Zeile nicht mit RUN zu starten!) als Listing 10.30 auch in diesem Buch den Quelltext.

Listing 10.30: Quelltext von »LOADER-MAKER 64«.

```
100  -.BA $0801
110  -.OB "LOADER-MAKER 64.P,W"
120  -;
130  -;
140  -; ******************************
150  -; *                            *
160  -; *  L O A D E R - M A K E R   *
170  -; *                            *
180  -; ******************************
190  -; *                            *
200  -; *   EIN PROGRAMMGENERATOR    *
210  -; *                            *
220  -; *    VON FLORIAN MUELLER     *
230  -; *                            *
240  -; ******************************
```

```
250  -;
260  -;
270  -;
280  -.GL BASIN  = $FFCF
290  -.GL SETPAR = $FFBA
300  -.GL SETNAM = $FFBD
310  -.GL LOAD   = $FFD5
320  -.GL READY  = $A474
330  -.GL NUMOUT = $BDCD
340  -.GL TASTPF = 631 ; TASTATURPUFFER
350  -.GL ANZAHL = 198 ; ENTHAELT ANZAHL
360  -;                  DER ZEICHEN IM
370  -;                  TASTATURPUFFER
380  -.GL KASSPF = 828 ; KASSETTENPUFFER
390  -;
400  -;
410  -.MA PRINT (TEXT)
420  -            LDA #<(TEXT)    ; MAKRO
430  -            LDY #>(TEXT)    ; FUER
440  -            JSR $AB1E       ; TEXTAUSGABE
450  -.RT
460  -;
470  -;
480  -;
490  -;
500  -.WO LINK+1 ; LINKPOINTER
510  -.WO 1985   ; ZEILENNUMMER
520  -.BY $9E    ; TOKEN FUER "SYS"
530  -           .TX "2061"
540  -LINK       .BY 0,0,0        ; ENDMARKIERUNG
550  -;                             DER BASIC-ZEILE
560  -;
570  -SYSTEM     LDX #0           ; FLAG FUER SYSTEM-
580  -           STX $9D          ; MELDUNGEN SETZEN
590  -;
600  -           LDX #$49         ; DEKR.-ZAEHLER
610  -SCHLEIFE1 LDA ABLAGE,X      ; LADEROUTINE
620  -           STA KASSPF,X     ; VON ABLAGE IN
630  -           DEX              ; DEN BEREICH
640  -           BPL SCHLEIFE1    ; KOPIEREN, IN
650  -;                             DEM SIE LAEUFT
660  -           JMP KASSPF       ; & STARTEN
670  -;
680  -;
690  -; ES FOLGT DIE LADEROUTINE, DIE HIER
700  -; AN FALSCHER STELLE ABGELEGT IST UND
710  -; VON DER "SCHLEIFE1" (600-640) IN
720  -; DEN ORIGINALBEREICH GESCHRIEBEN WIRD.
730  -;
740  -ABLAGE     LDA #1           ; FILENUMMER #1
750  -           TAY              ; SEKUNDAERADRESSE #1
760  -GERAETENR LDX #0           ; GERAETEADRESSE #?
770  -           JSR SETPAR       ; PARAMETER SETZEN
780  -;
790  -LAENGE     LDA #0           ; LAENGE DES FILENAMEN
800  -           LDX #<($35C)     ; ADRESSE DES
810  -           LDY #>($35C)     ; FILENAMEN: $035C
820  -           JSR SETNAM       ; NAMEN SETZEN
830  -;
840  -           LDA #0           ; FLAG FUER "LADEN"
850  -           JSR LOAD
860  -;
870  -FEHLER     BCS LOADERROR    ; LADEFEHLER?
880  -START      JMP 0            ; ZUR STARTADRESSE
```

```
890  -LOADERROR LDX #$1D        ; "LOAD ERROR"
900  -          JMP ($300)      ; AUSGEBEN
910  -;
920  -NAME      .BY 0,0,0,0     ; 16 BYTES
930  -          .BY 0,0,0,0     ; FUER FILENAMEN
940  -          .BY 0,0,0,0     ; RESERVIEREN
950  -          .BY 0,0,0,0
960  -;
970  -BASIC     STX $2D         ; POINTER FUER
980  -          STY $2E         ; PROGRAMMENDE SETZEN
990  -          JSR $E544       ; = PRINT CHR$(147)
1000 -          LDX #3          ; 3 BYTES IN
1010 -          STX ANZAHL      ; TASTATURPUFFER
1020 -;
1030 -SCHLEIFE2 LDA $0383,X     ; AUS DER TABELLE
1040 -          STA TASTPF,X    ; IN ZEILE 1100
1050 -          DEX             ; KOPIEREN
1060 -          BPL SCHLEIFE2
1070 -;
1080 -          JMP READY       ; WARMSTART
1090 -;
1100 -.BY "R",$D5,13 ; "R",SHIFT U,RETURN
1110 -;
1120 -; HIER ENDET DER PROGRAMMTEIL,
1130 -; DER MODIFIZIERT WIRD.
1140 -; ES FOLGT DIE MODIFIKATIONSROUTINE:
1150 -;
1160 -MDFIKATOR JSR $E544       ; = PRINT CHR$(147)
1170 -...PRINT (TEXT1)
1180 -; STARTADRESSE HOLEN
1190 -;
1200 -          JSR $AEFD       ; PRUEFT AUF KOMMA
1210 -          JSR $AD8A       ; HOLT PARAMETER
1220 -          JSR $B7F7       ; NACH $14/$15
1230 -;
1240 -          LDX $14         ; STARTADRESSE
1250 -          LDA $15         ; HOLEN,
1260 -          STX START+1     ; IM PROGRAMM
1270 -          STA START+2     ; ABLEGEN UND
1280 -          JSR NUMOUT      ; UND AUSGEBEN
1290 -;
1300 -;
1310 -; NUN WIRD NOCH DER ZU MODIFIZIERENDE
1320 -; PROGRAMMTEIL IN DEN AUSGANGSZUSTAND
1330 -; GEBRACHT:
1340 -;
1350 -          LDX #15         ; NAMEN MIT NULL-BYTES
1360 -          LDA #0          ; BELEGEN
1370 -SCHLEIFE3 STA NAME,X      ; DURCH EINE
1380 -          DEX             ; DEKREMENTIER-
1390 -          BPL SCHLEIFE3   ; SCHLEIFE
1400 -;
1410 -          STA SYSTEM+1    ; KEINE SYSTEMMELDUNGEN
1420 -;
1430 -          LDA #3          ; SPRUNGWEITE = 3
1440 -          STA FEHLER+1
1450 -;
1460 -          LDA #$A2        ; OPCODE FUER "LDX #"
1470 -          STA GERAETENR
1480 -;
1490 -;
1500 -; AN DIESER STELLE IST DAS "GERUEST"
1510 -; (DER ZU MODIFIZIERENDE TEIL)
1520 -; IM AUSGANGSZUSTAND
```

```
1530 -;
1540 -;
1550 -; EINGABE DES FILENAMEN
1560 -; ====================
1570 -;
1580 -...PRINT (TEXT2)
1590 -           LDX #0          ; ZAEHLER AUF 0
1600 -SCHLEIFE4 JSR BASIN
1610 -           CMP #13         ; ENDE DER EINGABE?
1620 -           BEQ WEITER1     ; JA->WEITER
1630 -           STA NAME,X      ; BYTE ABLEGEN
1640 -           INX
1650 -           CPX #16         ; 16 ZEICHEN MAX.
1660 -           BNE SCHLEIFE4   ; NAECHSTES ZEICHEN
1670 -;
1680 -; WENN DIESE STELLE DURCHLAUFEN WIRD,
1690 -; HAT DAS X-REGISTER DEN WERT 16.
1700 -;
1710 -; BEI "WEITER1" HINGEGEN KANN ES AUFGRUND
1720 -; DES BRANCH-BEFEHLS "BEQ WEITER1"
1730 -; UNTERSCHIEDLICHE WERTE HABEN.
1740 -;
1750 -WEITER1   STX LAENGE+1
1760 -;
1770 -;
1780 -; EINGABE DER GERAETEADRESSE
1790 -; -------------------------
1800 -;
1810 -...PRINT (TEXT3)
1820 -           JSR BASIN       ; HOLT ZEICHEN
1830 -           SEC             ; VOR SUBTRAKTION
1840 -           SBC #"0"        ; IM AKKU STEHT JETZT
1850 -;                            DIE ZAHL
1860 -;
1870 -           STA GERAETENR+1; ABLEGEN
1880 -           BNE WEITER2     ; GERAET<>0 : WEITER
1890 -; DA ALS GERAETENUMMER 0 EINGEGEBEN
1900 -; WURDE, MUSS DER GESAMTE BEFEHL
1910 -; "LDX #GERAET" IN "LDX $BA"
1920 -; UMGEWAENDELT WERDEN, DAMIT DAS
1930 -; NACHLADEN VON DEM GERAET ERFOLGT,
1940 -; VON DEM DER LADER EINGELESEN WIRD.
1950 -;
1960 -           LDA #$A6        ; OPCODE FUER "LDX ZP"
1970 -           STA GERAETENR
1980 -           LDA #$BA        ; "LDX $BA"
1990 -           STA GERAETENR+1; GENERIEREN
2000 -;
2010 -;
2020 -; MASCHINENPROGRAMM (J/N)?
2030 -; ----------------------
2040 -;
2050 -WEITER2   ... PRINT(TEXT4)
2060 -           JSR JANEIN      ; (JA/NEIN)?
2070 -           BEQ WEITER3     ; JA->WEITER
2080 -           LDA #$6C        ; SPRUNG AUF $036C
2090 -           LDY #$03        ; VERBIEGEN
2100 -           STA START+1     ; BEI $36C STEHT
2110 -           STY START+2     ; EINE ROUTINE,
2120 -;                            DIE DEN "RUN"-
2130 -;                            BEFEHL SIMULIERT
2140 -;
2150 -;
```

```
2160 -; SYSTEMMELDUNGEN (J/N)?
2170 -; ====================
2180 -;
2190 -WEITER3    ... PRINT(TEXT5)
2200 -            JSR JANEIN     ; (JA/NEIN)?
2210 -            BNE WEITER4    ; NEIN=>WEITER
2220 -            LDA #$80       ; FLAG FUER
2230 -            STA SYSTEM+1   ; SYSTEMMELDUNGEN
2240 -;
2250 -;
2260 -; LOAD ERROR  AUSGEBEN (J/N)
2270 -; ========================
2280 -;
2290 -;
2300 -WEITER4    ... PRINT(TEXT6)
2310 -            JSR JANEIN     ; (JA/NEIN)?
2320 -            BEQ WEITER5    ; NEIN=>WEITER
2330 -            LDA #0         ; FEHLERMELDUNGEN
2340 -            STA FEHLER+1   ; UNTERDRUECKEN
2350 -;
2360 -;
2370 -; PROGRAMMENDE
2380 -; ===========
2390 -;
2400 -;
2410 -WEITER5    ... PRINT(TEXT7)
2420 -;
2430 -; VEKTOR FUER BASIC-ENDE SETZEN
2440 -; =============================
2450 -;
2460 -;
2470 -            LDA #<(MDFIKATOR)
2480 -            STA $2D        ; LOW-BYTE
2490 -            LDA #>(MDFIKATOR)
2500 -            STA $2E        ; HIGH-BYTE
2510 -            JMP READY      ; SPRUNG INS BASIC
2520 -;
2530 -;
10000-;
10010-; ASCII-TABELLEN
10020-; =============
10030-;
10040-;
10050-TEXT1       .TX "LOADER-MAKER 64"
10060-            .BY 13,13
10070-            .TX "STARTADRESSE : "
10080-            .BY 0
10090-;
10100-TEXT2       .BY 13,13
10110-            .TX "FILENAME : "
10120-            .BY 0
10130-;
10140-TEXT3       .BY 13,13
10150-            .TX "GERAETENR. (1-9;0=UEBERNEHMEN) : "
10160-            .BY 0
10170-;
10180-TEXT4       .BY 13,13
10190-            .TX "MASCHINENPROGRAMM"
10200-            .BY 0
10210-;
10220-TEXT5       .BY 13,13
10230-            .TX "SYSTEMMELDUNGEN"
10240-            .BY 0
10250-;
```

```
10260-TEXT6      .BY 13,13
10270-           .TX "LOAD ERROR  AUSGEBEN"
10280-           .BY 0
10290-;
10300-TEXT7      .BY 13,13,18
10310-           .TX "*** LOADER GENERIERT ***"
10320-           .BY 13,13
10330-           .TX "MIT 'SAVE' SPEICHERN,"
10340-           .TX " MIT 'RUN' STARTEN"
10350-           .BY 0
10360-;
10370-TEXT8      .BY 13,13,18
10380-           .TX "*** PROGRAMMENDE ! ***"
10390-           .BY 13,13,0
10400-;
10410-TEXT9      .TX " (J/N)? "
10420-           .BY 0
10430-;
10440-;
20000-;
20010-; UNTERPROGRAMM FUER "J/N?"
20020-; ------------------------
20030-;
20040-;
20050-JANEIN     ... PRINT(TEXT9)
20060-           JSR BASIN      ; EINGABE HOLEN
20070-           CMP #"←"
20080-           BNE JANEIN1
20090-           PLA            ; SIEHE STAPEL-
20100-           PLA            ; MANIPULATION
20110-...PRINT(TEXT8)
20120-           JMP READY      ; SPRUNG INS BASIC
20130-JANEIN1    CMP #"J"       ; VERGLEICH MIT "J"
20140-           RTS            ; RUECKKEHR VOM
20150-;                           UNTERPROGRAMM
20160-.EN
```

Der LOADER-MAKER dient, der Name sagt es schon, zum Generieren von Ladeprogrammen, die ein anderes Maschinen- oder Basic-Programm nachladen und gleich starten sollen.

Anwendung von »LOADER-MAKER 64«

Da »Loader-Maker 64« alias »LISTING 10.31« auch für reine Anwendungszwecke interessant sein mag (ich selbst habe dieses Utility oft verwendet), sei zunächst eine Bedienungsanleitung gegeben.

Nach dem Laden von »LISTING 10.31« wird dieses Programm durch

```
SYS 2153,start     ←nicht mit RUN!
```

gestartet. Dabei ist »start« ein Parameter, der die Startadresse des nachzuladenden Programms ausdrückt. Soll ein Basic-Programm nachgeladen werden, hat diese Adresse keine Bedeutung (einfach SYS2153,0 eingeben!). Bei einem Maschinenprogramm handelt es sich hingegen um die Adresse, mit der das Programm über »SYS« gestartet wird (bei diesem Programm wäre es also 2153!).

Das Programm meldet sich mit »LOADER-MAKER 64« und gibt die Startadresse aus. Dazu können Sie den Filenamen eingeben.
Bei allen weiteren Eingaben (Gerätenummer, von der geladen werden soll; Maschinenprogramm j/n; Systemmeldungen wie »SEARCHING FOR« ausgeben j/n; LOAD ERROR bei Ladefehler ausgeben j/n) können Sie das Programm durch Eingabe des Linkspfeils abbrechen. Sind alle Eingaben getätigt worden, kommt die Meldung »LOADER GENERIERT«, und der Lader kann mit SAVE"NAME",8 abgespeichert werden.
Wenn das nachzuladende Programm von der Adresse geladen werden soll, von der auch das Ladeprogramm selbst eingegeben wurde, ist als Gerätenummer nur 0 einzugeben. Befassen wir uns nun mit dem Programm, dessen Qelltext Sie als Listing 10.30 bereits vorliegen haben.

Dokumentation von »LOADER-MAKER 64«

Die Zeilen 500 bis einschließlich 1100 stellen das Ladeprogramm in unmodifizierter Form dar und enthalten somit viele Dummywerte, wie zum Beispiel die an und für sich unsinnige Startadresse 0 in Zeile 880. Mit 1160 (Label MDFIKATOR) beginnt die Modifikationsroutine, welche ja über SYS2153,start aufgerufen wird. In 1280 wird die zuvor eingelesene Adresse »start« hinter dem Titel ausgegeben.

1260/1270 schreiben die Startadresse hinter den JMP-Befehl in Zeile 880.

1300–1500 bringen das mit Ausnahme der Startadresse unmodifizierte Gerüst in seinen Ausgangszustand, der dann nach Bedarf geändert wird.

1550–1770 holen den Filenamen, legen ihn bei NAME (Zeilen 920–950) ab, berechnen gleich die Länge des Filenamens und tragen diese bei LAENGE (790) ein.

1780–1990 holen die Geräteadresse. Da diese als Tastatureingabe zunächst im ASCII-Code vorliegt, muß der ASCII-Code von 0 abgezogen werden (1830/1840). Wurde 0 eingegeben, wird der Befehl »LDX #DEVICE« (760) in »LDX $BA« geändert; die Adresse $BA enthält nämlich jeweils die Geräteadresse, von welcher die letzte Datei gelesen wurde.

2000–2150 fragen, ob das nachzuladende Programm mit der per SYS übermittelten Startadresse gestartet wird (Eingabe »J«). Wurde »N« eingegeben, muß das Programm über den Basic-Befehl RUN gestartet werden. Auf eine entsprechende Routine (970–1080) wird die Startadresse gestellt, was die Zeilen 2080–2110 erledigen.

2160–2250 ermöglichen die Einstellung, ob »SEARCHING«, »LOADING« etc. ausgegeben werden sollen.

Soll im Falle eines Ladefehlers das Programm nicht gestartet und statt dessen »LOAD ERROR...« ausgegeben werden, wird dies bei 2260–2340 bestimmt. Bei Unterdrückung der

Fehlerausgabe wird einfach der BCS-Befehl aus Zeile 870 unschädlich gemacht, indem die Zeilen 2330/2340 die Sprungweite mit 0 beziffern.
Am Programmende wird noch eine Meldung ausgegeben (2410) und der Vektor für das Ende des Basic-Programms neu gesetzt (2470–2500), damit das generierte Ladeprogramm normal über SAVE gespeichert werden kann.

10000–10440 enthalten nur die Texttabellen.

Von 20000 bis zur letzten Zeile (20160) steht ein Unterprogramm, das bei jeder J/N-Entscheidung über »JSR JANEIN« aufgerufen wird. Es gibt den Text »(J/N) ?« aus (20050) und holt eine Eingabe (20060). Ist diese »J«, so ist nach dem Verlassen des Unterprogramms (20140) das Zero-Flag gesetzt, andernfalls ist es gelöscht. Wurde der Linkspfeil eingegeben, wird das Programm abgebrochen (20090/20100 sowie 20120) und eine entsprechende Meldung ausgegeben (20110).

Daran, daß der Abbruch über Stapelmanipulation erfolgt, sieht man deutlich, daß der »LOADER-MAKER 64« fast ausschließlich auf Programmiertricks dieses Kapitels 10 basiert. Hätten Sie ihn vorher problemlos analysieren können?

10.10 Mehr über relative Adressierung

So wie wir schon die Tücken der Zeropage-Adressierung zumindest teilweise in den Griff bekommen konnten, wollen wir uns mit der in vergleichbarer Weise leistungsstarken Relativ-Adressierung auseinandersetzen.

10.10.1 So vermeidet man JMP

Oft muß eine Stelle in einem Programm angesprungen werden, ohne daß erst eine Bedingung geprüft wird; man nennt dies einen »unbedingten Sprung«, wie ihn etwa der Assembler-Befehl JMP ermöglicht. Diese Sprungstelle ist nicht selten weniger als 128 Byte vom Sprungbefehl entfernt, könnte also relativ adressiert werden. Dennoch ist es in vielen Fällen möglich, einen BRANCH-Befehl zu verwenden, obwohl diese Kommandos eigentlich eine Bedingung (C=0 ...) prüfen.
Beispiel:

```
7050      BNE 7040
7052      JMP 708A
```

Kann ersetzt werden durch:

```
7050      BNE 7040
7052      BEQ 708A
```

Denn bei 7052 ist in jedem Fall das Z-Flag = 0 (dafür sorgt der Abfang-Befehl BNE), somit wird IMMER verzweigt. Dies gilt auch für folgende Konstellation:

```
7050        LDA #00
7052        BEQ 708A
```

Man könnte den BEQ-Befehl in diesem Zusammenhang auch als »Pseudo-Verzweigungsbefehl« bezeichnen, da die Bedingung gar nicht überprüft werden müßte (sie ist sowieso erfüllt). Der BRANCH-Befehl übertrifft den JMP-Befehl deutlich an Effektivität, da ein Byte weniger verbraucht wird. Im übrigen ist auch bei

```
7050        BVS 7040
7052        CLV
```

der CLV-Befehl überflüssig, solange vor 7052 der Befehl von 7050 verarbeitet wird. Genau die umgekehrte Befehlsfolge, »CLV:BVC....« setzt übrigens GEOS wiederholt ein, wodurch aber letztlich kein Byte gespart wird. Der Anlaß ist hier wohl eher die Relokatibilität (Verschiebbarkeit) der einzelnen Routinen.

10.10.2 Zugriff auf Befehle in »Umgebung«

Unter »Umgebung« wollen wir den Bereich um einen Programmteil verstehen, der über relative Adressierung ansprechbar ist. Da in diesem oft ähnliche Befehlsfolgen stehen wie im restlichen Programm, läßt sich hier durch gezielten Zugriff auf die »Umgebung« der Speicherplatzbedarf senken.

Ein Beispiel sehen Sie, allerdings in Basic, in Listing 7.2 ab Zeile 1000, wo der Programmteil 1020–1040 auch von 2020, 3020, 4020 und 5020 auf diese Weise genutzt wird. Diese Verhaltensweise ist »parasitär« im wahrsten Sinne dieses griechischen Wortes, welches ja ursprünglich »daneben liegend« heißt.

Im Basic-Interpreter und Betriebssystem gibt es zahllose Beispiele hierfür (siehe »C 64 für Insider«). Auch der bereits in 10.9 angesprochene Trick, Befehle durch BIT zu »verstecken«, findet hier intensive Anwendung.

Zur Übung könnten Sie noch versuchen, im Programm »TABELLEN-BEISPIEL« (Listing 10.11) die Menüroutine, insbesondere HOME, DOWN, UP und EXEC, wo beispielsweise wiederholt »STX MPT« steht, durch Zugriff auf »Umgebung« zu optimieren. Besonders interessant dürfte es auch sein, einige JMP-Befehle schrittweise durch BRANCHes zu ersetzen, weil z.B. nach »LDX #0« das Z-Flag gesetzt ist etc.

10.11 Diverse Tips und Kniffe zur optimalen Speichernutzung

Mit übermäßig viel RAM ist der C64 bestimmt nicht gesegnet. Bei vielen Anwendungen (zum Beispiel »Datenverarbeitung«, siehe das Programm »Master-Base«) braucht man wirklich das letzte Byte. Sie erhalten hier mehrere Tips, wie man den wenigen vorhandenen Speicher möglichst rationell verwenden kann.

10.11.1 Kassettenpuffer in Bildschirmspeicher

Zu den speicherplatzaufwendigsten Einrichtungen gehören die »Puffer«. Der Kassettenpuffer beispielsweise belegt den RAM-Bereich $033C–$03FB, auf den man somit oft verzichten muß. Hier wollen wir einfach den Kassettenpuffer in den Bildschirmspeicher (in Normaleinstellung: ab $0400) verlegen:

```
LDA #<($0400)
LDY #>($0400)
STA $B2
STY $B3
```

Da der Bildschirm beim Kassettenbetrieb ohnehin abgeschaltet wird, fällt dies nicht einmal auf, sofern man den Bildschirm nach Beenden des Kassettenbetriebs unverzüglich löscht.
Ebenso kann man andere Puffer (RS 232 usw.), für die es einen entsprechenden Zeiger gibt, problemlos nach $0400 verlegen, sofern sie nicht größer als 1000 Byte sind.

10.11.2 Das RAM ab $E000

Ein Problem für sich stellt das RAM ab $E000 (also unter dem Betriebssystem!) dar. Diesen Speicher kann man nur durch Bank-Switching nutzen, wobei man noch auf das Betriebssystem verzichten muß, solange der $E000-Bereich auf RAM geschaltet ist. Hier können Sie sich zunutze machen, daß der VIC auch ohne Ändern des Prozessor-Ports (Adresse $01) auf diesen RAM-Bereich zugreifen kann. Für Grafikbilder oder einen geänderten Zeichensatz ist der $E000-Bereich bestens geeignet. Dazu fällt mir spontan der Titel einer meiner Lieblingsfilme ein: Escape to VICtory!

10.11.3 Makros oder Unterprogramme

Mit dem Assembler »Hypra-Ass« von der Masterdiskette haben Sie ein mächtiges Werkzeug in der Hand, dessen besondere Stärke in der Verarbeitung von Makros liegt. Die Beispiele, vor allem »LOADER-MAKER«, versuchten bereits gezielt, diese Stärke des Hypra-Ass zu nutzen.

Dabei sollten Sie eine wichtige Regel bedenken: Makros belegen mehr Platz im Speicher als die einmalige Definition eines Unterprogramms, welches später immer wieder über »JSR«, also mit 3 Byte, aufzurufen ist.

10.11.4 Bits als Flags

Jedes Programm benötigt eine große Menge »Flags«. Meist belegt ein Flag genau 1 Byte, für dessen Inhalt es oft nur zwei mögliche Werte gibt: einen für »JA« und einen für »NEIN«. Für diese primitive Unterscheidungsform genügt aber auch 1 Bit, was nur ein Achtel soviel Speicher kostet wie 1 Byte.
Wenn Sie die Register von VIC und SID (Video- und Sound-Chip) ansehen, werden Sie feststellen, daß fast jedes Register mehrere Funktionen übernimmt, weil jedem Bit eine eigene Bedeutung zukommt. Würde der VIC oder SID hier statt auf Bits auf Bytes zugreifen,

1. wäre er langsamer und
2. würde der Speicherplatzaufwand für die Register sich vervielfachen.

Man sollte also bei Flags in eigenen Programmen nach demselben Prinzip vorgehen, jedem Bit eine eigene Bedeutung geben und nur die Bits prüfen:

```
BIT FLAG
```

Danach ist das N-Flag gesetzt, falls das Bit 7 in FLAG gesetzt ist, und das V-Flag, falls das Bit 6 gesetzt ist.
Die übrigen Flags erhält man über das Z-Flag im Prozessor-Statusregister mit Hilfe des Akkus. Angenommen, man möchte testen, ob Bit 0 in FLAG gesetzt ist oder nicht, so erledigt dies folgende Befehlssequenz:

```
LDA #01
BIT FLAG
BNE ???          ; (Bit gesetzt)
.
.
; (Bit nicht gesetzt)
```

Der BIT-Befehl ANDet den Inhalt des Akkus mit dem Wert der Speicherzelle FLAG. Möchte man Bit 1 testen, so ist der Befehl »LDA #01« zu ersetzen durch »LDA #02«, bei Bit 2 durch »LDA #04«, bei Bit 3 durch »LDA #08«, bei Bit 4 durch »LDA #10« und bei Bit 5 durch »LDA #20«; für Bit 6 und 7 ist der Vergleichswert irrelevant.

10.11.5 Selbstmodifikation als Sparmaßnahme

Durch Selbstmodifikationen können Flags bekanntlich vermieden werden. Aber auch unabhängig davon bietet diese raffinierte Programmiertechnik viele Gelegenheiten, Speicherplatz

zu sparen; die Steuerung einer Sprungtabelle beispielsweise belegt mit Selbstmodifikation weniger Speicher als ohne.
Auch die »Wegwerfmethode« ist sehr vorteilhaft, wenn auch nicht unkompliziert: Programmteile werden einmal abgearbeitet und dann (zum Beispiel durch Nachladen) überschrieben. Der Einsatz von »Daten-Packern« eignet sich somit in zweierlei Hinsicht: Zum einen können Programmteile, zum anderen Datentabellen durch Pack-Algorithmen synthetisch verkürzt werden. Und wenn man auch nicht zu eigenen Komprimieralgorithmen greift, so sollte man doch jede Datenstruktur möglichst effektiv austüfteln!

10.12 Programmierbeispiel für Einsteiger, Fortgeschrittene und Profis

Hier sei nun ein Beispielprogramm vorgestellt, anhand dessen jeder Leser noch einiges lernen kann:

- Programmierung von RESET-Schutz mit Neustart des Programms
- Aufheben von <RUN/STOP RESTORE>
- Erzeugung von Zufallszahlen mit Hilfe der CIA-Timer
- Simple Multiplikation durch wiederholte Addition
- Veränderung des Zeichensatzes, Bank-Switching
- Darstellung eines blinkenden Cursors bei GET ($FFE4)-Eingaben
- und außerdem macht es auch einigen Spaß, dieses Programm vorzuführen!

Das Programm befindet sich unter dem Filenamen »LISTING 10.32(O)« als startbares Basic-Programm mit zusätzlichem Vorspann, der hier nicht weiter kommentiert sei, auf Diskette. Der Quelltext ist nicht im Format des Hypra-Ass, sondern als herkömmliches Basic-Programm abgelegt.
Das Tolle daran ist, daß sich sowohl Anwendung als auch Programmierung aufgrund ausführlicher Informationstexte und Kommentare *von selbst* erklären! Sogar ohne Systemhandbuch werden Sie damit weitestgehend zurechtkommen, mit »C 64 für Insider« bleibt Ihnen kein Detail verborgen.
Einziger Hinweis: Der Quelltext in der gelisteten Form ist das reine Programm, welches zum Zeitpunkt der Ausführung ab $8000 im Speicher steht; der sogenannte Objektcode hingegen besteht aus einem Vorspann, der diese Routine für Basic-Starts zugänglich macht, indem er die Aufgabe erfüllt, das eigentliche Programm nach $8000 zu transportieren.

Listing 10.32: TICO-Quellcode

```
formatiertes listing  teil 1

100:   8000                    *=   $8000   ; startadresse $8000
101:   8000                     .opt p4
102:                   ;    ↑ nur formatiertes listing ausgeben
103:                   ;
104:                   ; **********************************
105:                   ; **********************************
106:                   ; ****                          ****
107:                   ; ****        t i c o           ****
108:                   ; ****        =======           ****
109:                   ; ****                          ****
110:                   ; ****          der             ****
111:                   ; **** elektronische hellseher  ****
112:                   ; ****                          ****
113:                   ; ****                          ****
114:                   ; **** by   florian mueller     ****
115:                   ; ****                          ****
116:                   ; ****                          ****
117:                   ; **** kommentierter quelltext  ****
118:                   ; ****                          ****
119:                   ; **********************************
120:                   ; **********************************
121:                   ;
122:                   ;
123:                   ;
124:                   ;
125:                   ; ---------------------------
126:                   ; -                         -
127:                   ; -   symboldefinitionen    -
128:                   ; -                         -
129:                   ; ---------------------------
130:                   ;
131:                   ;
132:   0002            flag     =    2
133:   00fa            anzahl   =    $fa
134:   00fb            honorar  =    $fb
135:   00fc            nummer   =    $fc
136:                   ;
137:                   ;
138:                   ;
139:                   ; -----------------------------
140:                   ; -                           -
141:                   ; -   m o d u l s t a r t     -
142:                   ; -                           -
143:                   ; -----------------------------
144:                   ;
145:                   ;
146:   8000 09 80 09            .wordstart,start
146:   8004 c3 c2 cd            .asc "CBM80"
147:                   ;
148:                   ;
149:                   ;
150:                   ; ---------------------------
151:                   ; -                         -
152:                   ; -   initialisierung       -
153:                   ; -                         -
154:                   ; ---------------------------
155:                   ;
156:                   ;
```

formatiertes listing teil 2

```
157:   8009 20 50 fd start    jsr  $fd50    ; arbeitsspeicher initialisieren
158:   800c 20 15 fd          jsr  $fd15    ; hardware- und i/o-vektoren setzen
159:   800f 20 a3 fd          jsr  $fda3    ; interrupt vorbereiten
160:   8012 20 5b ff          jsr  $ff5b    ; videoreset ausloesen
161:   8015 58                cli
161:   8016 20 bf e3          jsr  $e3bf    ; ram initialisieren
162:   8019 a9 fe             lda  #254
162:   801b 8d 19 03          sta  793
162:   801e a9 c1             lda  #193
162:   8020 8d 18 03          sta  792      ; nmi-routine kuerzen
163:   8023 d8                cld           ; dezimal-flag loeschen
164:   8024 78                sei           ; interrupt verhindern
165:   8025 ad 18 d0          lda  53272    ; zeichensatz
166:   8028 29 f0             and  #240     ; umschalten
167:   802a 18                clc
168:   802b 69 0c             adc  #12
169:   802d 8d 18 d0          sta  53272
170:   8030 a9 33             lda  #51      ; prozessorport aendern
171:   8032 85 01             sta  1        ; (bank-switching)
172:   8034 a0 00             ldy  #0       ; zeichensatz aus
173:   8036 84 fa             sty  $fa      ; dem
174:   8038 84 fc             sty  $fc      ; charactergenerator
175:   803a a9 d0             lda  #$d0     ; ab $d000
176:   803c 85 fb             sta  $fb      ; nach $3000
177:   803e a9 30             lda  #$30     ; kopieren
178:   8040 85 fd             sta  $fd      ; $d000 = dezimal 53248
179:   8042 b1 fa    loop1    lda  ($fa),y   ; $3000 = dezimal 12288
180:   8044 91 fc             sta  ($fc),y
181:   8046 c8                iny
182:   8047 d0 f9             bne  loop1
183:   8049 e6 fb             inc  $fb
184:   804b e6 fd             inc  $fd
185:   804d a5 fb             lda  $fb
186:   804f c9 e0             cmp  #$e0     ; "schon alle zeichen kopiert ?"
187:   8051 d0 ef             bne  loop1    ; nein, dann weiter
188:   8053 a9 37             lda  #55      ; prozessorport auf
189:   8055 85 01             sta  1        ; normalwert setzen
190:   8057 58                cli           ; interrupt wieder erlauben
191:                          ;
192:                          ;
193:                          ; ----------------------------
194:                          ; -                          -
195:                          ; - zeichensatz modifizieren -
196:                          ; -                          -
197:                          ; ----------------------------
198:                          ;
199:                          ;
200:   8058 a0 00             ldy  #0
201:   805a b9 71 81 loop2    lda  newchar,y
202:   805d 99 08 30          sta  12296,y
203:   8060 c8                iny
204:   8061 c0 d0             cpy  #endchar-newchar
205:   8063 d0 f5             bne  loop2
206:                          ;
207:                          ;
208:                          ; -----------------------------
209:                          ; -                           -
210:                          ; - beginn des hauptprogramms -
```

```
formatiertes listing  teil 3

211:                    ; -                           -
212:                    ; -----------------------------
213:                    ;
214:                    ;
215:  8065 a9 07        init      lda  #7      ; gelber
216:  8067 8d 20 d0               sta  53280   ; rahmen
217:  806a 8d 21 d0               sta  53281   ; & hintergrund
218:  806d a9 08                  lda  #8      ; shift/c= verhindern
219:  806f 20 d2 ff               jsr  $ffd2
220:  8072 a9 00                  lda  #0      ; schwarz
221:  8074 8d 86 02               sta  646     ; als zeichenfarbe
222:  8077 20 44 e5               jsr  $e544   ; bildschirm loeschen
223:                    ;
224:                    ;
225:                    ;
226:                    ; -------------------------------
227:                    ; -                             -
228:                    ; -   ausgabe der erklaerung    -
229:                    ; -                             -
230:                    ; -------------------------------
231:                    ;
232:                    ;
233:  807a a9 41                  lda  #<text1
234:  807c a0 82                  ldy  #>text1
235:  807e 20 1e ab               jsr  $able
236:  8081 a9 bd                  lda  #<text2
237:  8083 a0 82                  ldy  #>text2
238:  8085 20 1e ab               jsr  $able
239:  8088 a9 4a                  lda  #<text3
240:  808a a0 83                  ldy  #>text3
241:  808c 20 1e ab               jsr  $able
242:                    ;
243:                    ;
244:                    ;
245:                    ; ------------------------------
246:                    ; -                            -
247:                    ; -  anzahl der fragen holen   -
248:                    ; -                            -
249:                    ; ------------------------------
250:                    ;
251:                    ;
252:                    ;
253:  808f a9 00                  lda  #0
254:  8091 85 cc                  sta  204     ; cursor einschalten
255:  8093 20 e4 ff get           jsr  $ffe4   ; holt zeichen in akku
256:  8096 f0 fb                  beq  get     ; keine taste => nochmal
257:  8098 c9 30                  cmp  #"0"    ; pruefung,
258:  809a 90 f7                  bcc  get     ; ob die gedrüeckte
259:  809c c9 3a                  cmp  #"9"+1  ; taste eine
260:  809e b0 f3                  bcs  get     ; zifferntaste ist
261:  80a0 a2 00                  ldx  #0      ; cursorblinken aus
262:  80a2 86 cf                  stx  207
263:  80a4 e8                     inx          ; cursor aus
264:  80a5 86 cc                  stx  204
265:  80a7 20 d2 ff               jsr  $ffd2   ; ziffer ausgeben
266:  80aa 38                     sec
267:  80ab e9 30                  sbc  #"0"    ; zahl berechnen
268:  80ad 85 fa                  sta  anzahl
```

```
formatiertes listing  teil 4

269:    80af d0 03              bne  next1
270:    80b1 a9 ff              lda  #255     ; unendlich viele fragen
271:    80b3 2c                 .byte$2c      ; versteckter bit-befehl
272:    80b4 a9 00     next1    lda  #0       ; begrenzte anzahl von fragen
273:    80b6 85 02              sta  flag
274:                   ;
275:                   ;
276:                   ;
277:                   ; --------------------------------
278:                   ; -                              -
279:                   ; - berechnung des honorars nach -
280:                   ; -                              -
281:                   ; - der formel honorar=fragen*5  -
282:                   ; -                              -
283:                   ; --------------------------------
284:                   ;
285:                   ;
286:                   ;
287:    80b8 a6 fa              ldx  anzahl
288:    80ba a9 00              lda  #0
289:    80bc 18        loop3    clc
290:    80bd 69 05              adc  #5
291:    80bf ca                 dex
292:    80c0 d0 fa              bne  loop3
293:    80c2 85 fb              sta  honorar
294:                   ;
295:                   ;
296:                   ;
297:                   ; ------------------------------
298:                   ; -                            -
299:                   ; -  kostenvoranschlag  machen -
300:                   ; -                            -
301:                   ; ------------------------------
302:                   ;
303:                   ;
304:                   ;
305:    80c4 a5 02              lda  flag     ; "unendlich viele fragen?"
306:    80c6 d0 18              bne  next2    ; ja, dann kein kostenvoranschlag
307:    80c8 a9 91              lda  #<text4
308:    80ca a0 83              ldy  #>text4
309:    80cc 20 1e ab           jsr  $ab1e
310:    80cf a9 00              lda  #0
311:    80d1 a6 fb              ldx  honorar
312:    80d3 20 cd bd           jsr  $bdcd    ; honorar ausgeben
313:    80d6 a9 ae              lda  #<text5
314:    80d8 a0 83              ldy  #>text5
315:    80da 20 1e ab           jsr  $ab1e
316:    80dd 4c e7 80           jmp  next3
317:    80e0 a9 ba     next2    lda  #<text6
318:    80e2 a0 83              ldy  #>text6
319:    80e4 20 1e ab           jsr  $ab1e
320:    80e7 a9 03     next3    lda  #<text7
321:    80e9 a0 84              ldy  #>text7
322:    80eb 20 1e ab           jsr  $ab1e
323:    80ee a9 00              lda  #0       ; cursor einschalten
324:    80f0 85 cc              sta  204
325:    80f2 20 e4 ff get1      jsr  $ffe4    ; taste holen
326:    80f5 f0 fb              beq  get1
```

```
formatiertes listing  teil 5

327:    80f7 a2 00          ldx  #0
328:    80f9 86 cf          stx  207
329:    80fb e8             inx
330:    80fc 86 cc          stx  204
331:    80fe a5 02          lda  flag     ; "unendlich viele fragen?"
332:    8100 f0 03          beq  next4    ; nein, dann weiter
333:    8102 4c 65 80       jmp  init     ; ja, also neustart
334:    8105 a9 20   next4  lda  #" "     ; space ausgeben
335:    8107 20 d2 ff       jsr  $ffd2
336:    810a 20 44 e5       jsr  $e544    ; bildschirm loeschen
337:                 ;
338:                 ;
339:                 ; ------------------------------
340:                 ; -                            -
341:                 ; -  eingabe und beantwortung  -
342:                 ; -                            -
343:                 ; -         der fragen         -
344:                 ; -                            -
345:                 ; ------------------------------
346:                 ;
347:                 ;
348:                 ;
349:    810d a9 01          lda  #1       ; zaehler fuer nummer der frage
350:    810f 85 fc          sta  nummer   ; auf 1 setzen
351:    8111 a9 2b   loop4  lda  #<text8
352:    8113 a0 84          ldy  #>text8
353:    8115 20 1e ab       jsr  $able
354:    8118 a9 00          lda  #0
355:    811a a6 fc          ldx  nummer
356:    811c 20 cd bd       jsr  $bdcd    ; nummer der frage ausgeben
357:    811f a9 41          lda  #<text9
358:    8121 a0 84          ldy  #>text9
359:    8123 20 1e ab       jsr  $able
360:    8126 20 cf ff       jsr  $ffcf    ; frage einlesen
361:    8129 ad 05 dc       lda  $dc05    ; zufallszahl
362:    812c 6d 04 dc       adc  $dc04    ; zwischen 0 und 3
363:    812f 29 03          and  #3       ; in akku holen
364:    8131 aa             tax           ; und ins x-register schreiben
365:    8132 bd 4d 84       lda  low,x
366:    8135 bc 51 84       ldy  high,x
367:    8138 20 1e ab       jsr  $able    ; antwort ausgeben
368:    813b a5 fc          lda  nummer   ; pruefen,
369:    813d c5 fa          cmp  anzahl   ; ob schon alle
370:    813f 08             php           ; fragen gestellt wurden
371:    8140 e6 fc          inc  nummer   ; nummer erhoehen
372:    8142 28             plp           ; status nach vergleich holen
373:    8143 90 cc          bcc  loop4    ; noch nicht alle fragen, also weiter
374:    8145 a9 bb          lda  #<tfor1
375:    8147 a0 84          ldy  #>tfor1
376:    8149 20 1e ab       jsr  $able
377:    814c a5 fb          lda  honorar  ; honorarforderung
378:    814e 18             clc
378:    814f 69 0a          adc  #10      ; um 10 erhoehen (!)
379:    8151 aa             tax           ; und ausgeben
380:    8152 a9 00          lda  #0
381:    8154 20 cd bd       jsr  $bdcd
382:    8157 a9 e0          lda  #<tfor2
383:    8159 a0 84          ldy  #>tfor2
```

```
formatiertes listing  teil 6

384:   815b 20 1e ab          jsr  $able
385:   815e a9 00             lda  #0
386:   8160 85 cc             sta  204
387:   8162 20 e4 ff get2     jsr  $ffe4
388:   8165 f0 fb             beq  get2
389:   8167 a2 00             ldx  #0
390:   8169 86 cf             stx  207
391:   816b e8                inx
392:   816c 86 cc             stx  204
393:   816e 4c 65 80          jmp  init     ; neustart
394:                 ;
395:                 ;
396:                 ;
397:                 ; ------------------------------
398:                 ; -                            -
399:                 ; -   codes der neuen zeichen  -
400:                 ; -                            -
401:                 ; ------------------------------
402:                 ;
403:                 ;
404:                 ;
405:                 ;
406:   8171 00 3c 3c newchar  .byte0,60,60.102,126,126,230,230
407:   8179 00 38 38          .byte0,56,56,108,124,102,254,254
408:   8181 00 68 fc          .byte0,104,252,196,192,196,252,104
409:   8189 00 3c 3e          .byte0,60,62,230,102,102,254,252
410:   8191 00 7e 7e          .byte0,126,126,98,120,98,254,254
411:   8199 01 fe fe          .byte1,254,254,166,56,56,240,240
412:   81a1 00 3e 7e          .byte0,62,126,102,104,126,254,254
413:   81a9 00 e6 66          .byte0,230,102,102.104,126,254,254
414:   81b1 00 30 30          .byte0,48.48,48,48,48,112,112
415:   81b9 00 fc fc          .byte0,252,252,12,12,204,124,120
416:   81c1 00 e6 e6          .byte0,230,230,108,120,124,138,230
417:   81c9 00 e0 e0          .byte0,224,224,96,96,102,126,124
418:   81d1 00 c4 ee          .byte0,196,238,254,214,198,198,198
419:   81d9 00 e4 e6          .byte0,228,230,246,254,206,206,198
420:   81e1 00 3c 3c          .byte0,60,60,102,102.102,60,60
421:   81e9 00 3c 7e          .byte0,60,126,102,126,124,192,192
422:   81f1 00 3c 3c          .byte0,60,60,102,102,110,60,63
423:   81f9 00 3c 7e          .byte0,60,126,102,126,124,204,198
424:   8201 00 1e 1e          .byte0,30,30,26,24,152,248,248
425:   8209 00 fe fe          .byte0,254,254,178,48,48,112,112
426:   8211 00 e6 e6          .byte0,230,230,102,102,102,126,126
427:   8219 00 e6 e6          .byte0,230,230,102,48,48,12,12
428:   8221 00 c6 c6          .byte0,198,198,198,214,254,238,106
429:   8229 00 81 ee          .byte0,129,238,108,16,108,238,198
430:   8231 00 81 cc          .byte0,129,204,112,48,48,48,48
431:   8239 00 fe fe          .byte0,254,254,14,48,224,254,254
432:   8241          endchar  =    *
433:                 ;
434:                 ;
435:                 ; ------------------------------
436:                 ; -                            -
437:                 ; -          t e x t e         -
438:                 ; -                            -
439:                 ; ------------------------------
440:                 ;
441:                 ;
```

```
formatiertes listing  teil 7

442:   8241 48 41 4c text1    .asc "hallo , ich bin tico , der elektronische"
443:   8269 0d                .byte13
444:   826a 48 45 4c          .asc "hellseher. ich wurde von florian mueller"
445:   8292 0d                .byte13
446:   8293 46 55 45          .asc "fuer s+s soft programmiert. erfunden hat"
447:   82bb 0d 00             .byte13,0
448:   82bd 4d 49 43 text2    .asc "mich b. goltzsch."
449:   82ce 0d 0d             .byte13,13
450:   82d0 49 43 48          .asc "ich kann ihnen  auf jede frage eine ant-"
451:   82f8 0d                .byte13
452:   82f9 57 4f 52          .asc "wort geben.die so gestellt ist, dass man"
453:   8321 0d                .byte13
454:   8322 4d 49 54          .asc "mit 'ja' oder 'nein' antworten kann."
455:   8346 0d 0d 0d          .byte13,13,13,0
456:   834a 57 49 45 text3    .asc "wieviele fragen wollen sie stellen ?"
457:   836e 0d 0d             .byte13,13
458:   8370 28 31 2d          .asc "(1-9/ 0 fuer unendlich viele) : "
459:   8390 00                .byte0
460:   8391 0d 0d    text4    .byte13,13
461:   8393 44 41 53          .asc "das honorar wird etwa bei "
462:   83ad 00                .byte0
463:   83ae 20 44 4d text5    .asc " dm liegen."
464:   83b9 00                .byte0
465:   83ba 0d 0d    text6    .byte13,13
465:   83bc 53 49 45          .asc "sie glauben doch nicht im ernst,dass ich"
466:   83e4 0d                .byte13
467:   83e5 4d 49 52          .asc "mir den mund fusselig rede ??"
468:   8402 00                .byte0
469:   8403 0d 0d 0d text7    .byte13,13,13
469:   8406 20 20 20          .asc "    (rvs)bitte druecken sie eine taste(off)
"
470:   842a 00                .byte0
471:   842b 0d 0d    text8    .byte13,13
472:   842d 90 08 8e          .asc "(blk)(dish)(swuc)wie lautet ihre "
473:   8440 00                .byte0
474:   8441 2e 20 46 text9    .asc ". frage ?"
475:   844a 0d 0d 00          .byte13,13,0
476:   844d 55 5b 7b low      .byte<k0,<k1,<k2,<k3
477:   8451 84 84 84 high     .byte>k0,>k1,>k2,>k3
478:                  ;
479:   8455 0d 0d    k0       .byte13,13
480:   8457 4a 41 2e          .asc "ja."
481:   845a 00                .byte0
482:   845b 0d 0d    k1       .byte13,13
483:   845d 4e 45 49          .asc "nein, das ist nicht moeglich."
484:   847a 00                .byte0
485:   847b 0d 0d    k2       .byte13,13
486:   847d 48 4f 45          .asc "hoechstwahrscheinlich."
487:   8493 00                .byte0
488:   8494 0d 0d    k3       .byte13,13
489:   8496 44 41 53          .asc "das halte ich fuer unwahrscheinlich."
490:   84ba 00                .byte0
491:   84bb 0d 0d 0d tfor1    .byte13,13,13
492:   84be 42 49 54          .asc "bitte legen sie mein honorar von "
493:   84df 00                .byte0
494:   84e0 20 44 4d tfor2    .asc " dm"
495:   84e3 0d 0d             .byte13,13
496:   84e5 41 55 46          .asc "auf den monitor. druecken sie dann eine"
497:   850c 0d 0d             .byte13,13

formatiertes listing  teil 8

498:   850e 54 41 53          .asc "taste. "
499:   8515 00                .byte0
]8000-8516
no errors
```

11 Utilities zum Diskettenlaufwerk

Dieses Kapitel beschreibt eine Zusammenstellung der wichtigsten und vielfältigsten Hilfsprogramme, die die Arbeit mit Disketten und Floppy-Laufwerken unterstützen. Der überwiegende Teil der Programme stammt aus dem 64'er-Magazin und ist somit tausendfach in der Praxis bewährt. Dieses Kriterium war auch bei der Auswahl entscheidend.
Dennoch müssen Sie sehr vorsichtig mit allen Programmen umgehen, die in irgendeiner Weise Veränderungen auf einer Diskette vornehmen; arbeiten Sie dabei nie mit der beiliegenden Masterdiskette oder einer eigenen Programmdiskette, sondern nur mit Sicherheitskopien, die notfalls zerstört werden dürfen!

11.1 Auf das »!« kommt es an

Geht es Ihnen auch so: Man weiß nie genau, ob das Programm, das man laden will, jetzt »Disk Copy V 1.0« oder »Disk Copy V1.0« heißt. Versucht man es mit »Disk*«, so lädt man nach Murphys Gesetz natürlich »Disk-Monitor«. Es hilft also nichts: Man lädt das Directory, listet es und ärgert sich, weil das Programm, das man eigentlich laden wollte, ganz oben steht und beim Scrollen verschwindet. Also nochmal »LIST«, dann mit dem Cursor in die richtige Reihe fahren, »LOAD« eintippen, Cursor hinter den Programmnamen, »,8« eintippen. Der Computer antwortet mit einem verächtlichen »SYNTAX ERROR«, weil man wieder einmal den Doppelpunkt vergessen hat.
Vielleicht ist dieses Beispiel für Besitzer von Floppy-Speedern längst nicht mehr Realität, für alle anderen Anwender hingegen dürfte es durchaus der Praxis entsprechen. Aus diesem Gedanken schufen Norfried Mann und Dietrich Weineck, zwei bekannte 64'er-Programmautoren, das Programm »!«.
Es wird mit

```
LOAD"!",8,1
```

geladen, startet automatisch, liest das Directory ein und schreibt hinter jedes Programm eine Nummer. Wenn der Bildschirm voll ist, wartet es auf Ihre Eingabe. Tippen Sie »+«, so bekommen Sie die nächsten Programme angezeigt. Tippen Sie aber die Programmnummer und dahinter ein »L«, wird das entsprechende Programm geladen. Handelt es sich um ein Programm, das absolut geladen werden muß (mit »,8,1«), tippt man hinter die Nummer ein »A«.

Es empfiehlt sich, dieses wirklich hilfreiche Tool auf jede Diskette zu kopieren. Eine äußerst aufschlußreiche Beschreibung, aus der jeder Maschinenprogrammierer eine Menge lernen kann und viele andere bereits gelernt haben (auch der Autor dieses Buches), finden Sie in 64'er 12/84 auf Seite 92.

11.2 Floppy-Lister

Mit einem einfachen SYS-Aufruf können Sie Programme und sequentielle Dateien direkt von Diskette listen. Programme im Speicher bleiben dabei erhalten.
»Floppy-Lister« bietet zwei Möglichkeiten, ein Programm von Diskette zu listen. Basic-Programme und sequentielle Dateien können entweder als Klartext oder als Speicherauszug (Dumps) gelistet werden. Im Dumpmodus werden sämtliche Daten als hexadezimale Codes ausgegeben und, soweit möglich, in ASCII-Zeichen übersetzt. Deshalb eignet er sich besonders zur Analyse unbekannter Programmdateien: Maschinenprogramme dürfen grundsätzlich nur auf diese Art gelistet werden, will man einen Absturz vermeiden.
Der »Floppy-Lister« kann direkt geladen werden (LOAD"FLOPPYLISTER",8,1) und mit SYS49152,»XY:filename« aufgerufen werden. Vergessen Sie nach dem Laden nicht, den NEW-Befehl einzugeben, um die Basic-Zeiger in einen vernünftigen Zustand zu bringen. »X« steht für Fileart: entweder »P« für Programm oder »S« für sequentielle Datei. »Y« steht für den Modus, in dem gelistet werden soll: »L« für einfaches Listen und »D« für Ausgabe in Form eines Speicher-Dumps.
Zum Beispiel listet der Befehl

```
SYS 49152,"SL:TEST"
```

eine sequentielle Datei mit dem Namen »TEST«. Abkürzungen des Programmnamens durch »*« bzw. »wild cards« mit »?« sind zulässig; falsche Eingaben werden mit einer Fehlermeldung quittiert.
Mit <CTRL> kann die Ausgabe verlangsamt und mit <RUN/STOP> unterbrochen werden. Eine Fortsetzung erfolgt mit <A>, ein vorzeitiger Abbruch des Listings mit <DEL>. Am Ende eines Listings ist immer <Leertaste> zu betätigen.
Das Maschinenprogramm für »Floppy-Lister« liegt im Bereich von $C000-$C2B2 (49152-49842). An das Programm schließt sich noch ein Pufferbereich an, wo Daten zwischengespeichert werden. »Floppy-Lister« läuft mit vielen Erweiterungen.

11.3 Super-Autostart

Das selbsttätige Starten von Programmen »während des Ladens« fasziniert immer wieder. Darauf haben Sie schon lange gewartet: auf einen Autostart-Generator, der viele sinnvolle Eigenschaften aufweist. Dazu gehören:

- kurzes Programm (sowohl der Generator als auch das Ergebnis selber)
- einfache Anwendung
- Schutz gegen <RUN/STOP RESTORE> und Reset für fertige Programme
- Kodier-/Dekodier-Funktion

Zum Aktivieren eines Programms laden Sie das Programm »AUTOSTART« über LOAD "AUTOSTART",8,1, geben Sie NEW ein und holen Sie schließlich das Programm in den Speicher, welches Sie mit dem Autostart versehen möchten.
Geben Sie nun folgenden Befehl ein:

```
SYS 49152,code,"Haupt-Name","Lader-Name"
```

»Code« ist eine beliebige Zahl zwischen 0 und 255 für die Verschlüsselung. »Haupt-Name« und »Lader-Name« sind die zukünftigen Namen des kodierten Hauptteils bzw. des Lade-Programms auf der Diskette. Der Lader ist später mit »,8,1« in den C64 zu lesen. Läßt man beim SYS-Befehl »Lader-Name« weg, wird nur der kodierte Hauptteil neu gespeichert, sofern man Änderungen an diesem vornehmen wollte. Wichtig ist dann nur, daß die Codezahl des Hauptprogramms mit der des Laders übereinstimmt. Im Zweifelsfall sollte man lieber den alten Lader löschen und beide Teile neu generieren.

11.4 Kompakt durch Komprimierung

Fast alle Programme, egal ob Basic oder Assembler, weisen Folgen von gleichen Zeichen auf. Denken Sie nur an Grafikbilder, die in einem Programm enthalten sein könnten. Diese Zeichenfolgen lassen sich kürzen.
Damit die gekürzte Version lauffähig ist, wird vor das eigentliche Programm noch ein »Entpacker« gesetzt, der auch mit abgespeichert wird. Von jedem Programm bleibt nach Komprimieren nur noch eine Zeile mit einem SYS-Befehl zum Start des Entpackers übrig. Tippen Sie RUN ein, wird vor dem eigentlichen Programmstart das ursprüngliche Programm aus den komprimierten Daten wieder zusammengesetzt, was sich durch eine kurze »Pause« bemerkbar macht. Das komprimierte Programm wird dazu byteweise wieder an die Stelle gesetzt, wo es vor der Behandlung mit dem Komprimierungsprogramm stand.
Stoppen Sie ein Basic-Programm nach dem Entpacken mit <RUN/STOP> oder <RUN/STOP RESTORE>, sehen Sie wieder das normale Basic-Programm in seiner vollen Länge.
Sie erhalten nun zwei Programme, von denen das zweite in den meisten Fällen leistungsfähiger ist, man aber oft auch das erste benötigt.

11.4.1 File-Compressor

Dieses Programm kürzt Programme, sofern sie mindestens eine Basic-Zeile haben und somit über RUN gestartet werden, auf bis zu 50 Prozent ihrer Länge.

1. LOAD "FILE-COMPRESSOR",8
2. RUN
3. Diskette mit dem zu kürzenden Programm einlegen
4. Namen des Programms eingeben und <RETURN> drücken, um das Programm zu laden
5. Diskette mit genügend freiem Platz einlegen
6. Namen des gekürzten Programmes eingeben und <RETURN> drücken, um die gekürzte Version zu speichern

11.4.2 File-Compactor

Der »FILE-COMPACTOR« kürzt beliebige Programme um bis zu 55 Prozent. Dadurch wird jedes Programm mit einer Länge von mindestens 25 Blöcken auf Diskette erheblich gekürzt. Das Programm, das gekürzt werden soll, muß ab dem Basic-Start (Adresse $0801) in den Speicher geladen werden und darf nicht länger als 48639 Byte sein. Nach dem Kürzen steht eine SYS-Zeile am Anfang, die den Entpacker startet, welcher – Sie wissen es schon – das ursprüngliche Programm im Speicher herstellt und startet.

1. LOAD "FILE-COMPACTOR",8
2. RUN
3. Jetzt liegt der Compactor im richtigen Bereich
4. Programm laden
5. SYS 51000 zum Start des Compactor
6. Anhand der ausgegebenen Zahl eingesparter Bytes entscheiden, ob man speichern will; falls ja:
7. SYS 50965»NAME« ersetzt SAVE"NAME",8; mit SAVE könnten nicht alle Programme zufriedenstellend gesichert werden

11.5 Disketten-Ökonomie

Unerwünschte Leerräume auf der Diskette (»128 BLOCKS FREE«) ? Durch die richtige Zusammenstellung der Dateien kann dies vermieden werden. Der Disk-Optimizer (Listing 11.1) erledigt dies für Sie.

Listing 11.1: DISK-OPTIMIZER, Filename "DISK-OPTIM. II"

```
10 S=664
   MA=50
   BA=1184
   AZ=0
   G=0
   GE=0
   K=0
   Z=0
   X=0
   Y=0
   V=0
20 DIMF%(MA)
   GOTO80
30
40 PRINT"(CLR)DISK-OPTIMIZER II"
   PRINT"──────────────────"
50 PRINT"BY MARKUS NAEHER(DOWN)"
   RETURN
60
70 REM  *** EINLESEN ***
80 GOSUB40
   PRINT"LAENGEN DER PROGRAMME EINGEBEN(DOWN)"
   FORX=0TOMA
90 PRINTX+1"(LEFT). PROGRAMM ";
   INPUTF%(X)
   IFF%(X)>=STHENPRINT"(UP)(UP)"
   GOTO90
100 IFF%(X)<0THENX=X-1
    PRINT"(UP)(UP)(UP)"
    GOTO90
110 IFF%(X)<>0THENNEXT
120 AZ=X-1
130
140 REM  *** VON GROSS NACH KLEIN ORDNEN ***
150 FORX=0TOAZ-1
    G=0
    FORY=XTOAZ
    IFF%(Y)>GTHENG=F%(Y)
    GE=Y
160 NEXT
    F%(GE)=F%(X)
    F%(X)=G
    NEXT
170 GOSUB40
    FORX=0TOAZ
    PRINT" "F%(X),
    NEXT
    PRINT"(DOWN)
180
190 REM  *** SUCHE NACH KOMBINATIONEN ***
200 OPEN1,4
    PRINT#1,"DISK-OPTIMIZER II"
    DIMK%(AZ*10,AZ),P%(AZ)
210 FORZ=1TOAZ-1
    FORX=0TOZ
    P%(X)=X
    NEXT
    V=Z
220 REM ** AUSGABE DER MOMENTANEN KONFIGURATION **
230 FORX=P%(V)-1TOAZ
    POKEBA+10*X,32
    NEXT
```

```
    FORX=ØTOZ
    POKEBA+10*P%(X),42
    NEXT
240 REM ** TESTEN, OB AKTUELLE KOMBINATION ALS SUMME S ERGIBT, DANN AUSGABE **
250 G=Ø
    FORX=ØTOZ
    G=G+F%(P%(X))
    NEXT
    PRINT"SUMME
    "G"    "
    IFG<>SGOTO27Ø
260 FORX=ØTOZ
    K%(K,P%(X))=1
    PRINT#1,F%(P%(X));
    NEXT
    PRINT#1
    K=K+1
    GOTO29Ø
270 IFG<SANDV=ØGOTO32Ø
280 REM ** NAECHSTE KOMBINATION ERMITTELN **
290 FORX=ZTOØSTEP-1
    IFP%(X)=AZ-Z+XTHENNEXT
    GOTO32Ø
300 V=X
    P%(V)=P%(V)+1
    IFX<ZTHENFORX=V+1TOZ
    P%(X)=P%(X-1)+1
    NEXT
310 GOTO23Ø
320 G=Ø
    FORX=ØTOZ+1
    G=G+F%(AZ-X)
    NEXT
    IFG<STHENNEXTZ
330 IFG=STHENFORX=AZ-Z-1TOAZ
    K%(K,X)=1
    PRINT#1,F%(X);
    NEXT
    PRINT#1
    K=K+1
340

350 REM *** AUSGABE DER KOMBINATIONEN ALS TABELLE ***
360 IFK=ØTHENCLOSE1
    END
370 FORX=ØTOAZ
    PRINT#1,RIGHT$("          "+STR$(F%(X)),LEN(STR$(F%(Ø))));
380 FORY=ØTOK-1
    PRINT#1,CHR$(46-4*K%(Y,X));
    NEXT
    PRINT#1
    NEXT
```

Nach dem Start werden die Längen der Programme eingegeben. Programme, die aus mehreren Teilprogrammen bestehen, sollten als einziges Programm eingegeben werden, damit sie auf die gleiche Diskette kommen; die Längen der Teildateien müssen zusammengezählt werden. Wird eine negative Zahl eingegeben, so kann die letzte Eingabe nochmals korrigiert werden. Die Eingabe von »0« beendet den Programmteil. Aus der vom Programm erstellten Tabelle ist sofort ersichtlich, welche Kombinationen sich überschneiden. Dadurch läßt es sich damit besser weiterarbeiten als mit bloßen Zahlenreihen.

11.6 Disc-Wizard

Zuerst einmal sei betont, daß Sie zur Verwendung dieses Programms eine Commodore-Floppy 1541 (nicht 1570/1571!) besitzen müssen. Mit GEOS-Disketten ist das Programm nur dann verwendbar, wenn Sie ausreichend über das GEOS-Diskettenformat informiert sind. Alle nötigen Informationen und Hinweise finden Sie in »C64 – Alles über GEOS 1.2« oder »C64 – Alles über GEOS 1.3«. Noch eine Warnung: Zum Austesten der einzelnen Funktionen des Disc-Wizard sollten Sie unbedingt eine Diskette mit unwichtigem Inhalt nehmen. Denn mit dem Diskettenmonitor könnten Sie unter Umständen Blöcke mit wichtigen Daten restlos zerstören!
Nach dem Start mit RUN hören Sie einen Signalton, und es erscheint das Hauptmenü; falls Sie zu diesem Zeitpunkt nicht die Floppy eingeschaltet haben, erscheint die Meldung »No Connection With Floppy«, und das Programm wartet darauf, daß Sie Ihr Laufwerk einschalten und dies per Tastendruck bestätigen. Vor dem Menüpunkt »Directory« steht ein reverses Kästchen mit einem beweglichen Strich; dies ist Ihr »Cursor« zur Anwahl der Funktionen. Mit <CRSR DOWN> bewegen Sie die Markierung nach unten, mit <CRSR UP> oder <CRSR RIGHT> nach oben.
Das Hauptmenü besteht aus zwei »Bildschirmfenstern«, zwischen denen Sie mit <F7>, <F5> oder der Leertaste hin- und herschalten können. <RETURN> startet die gewählte Funktion. Im unteren Bildschirmbereich wird ständig der »Status«, also der Fehlerkanal der Floppy, angezeigt.
Folgende Menüpunkte stehen zur Auswahl:

DIRECTORY
Funktion: Einlesen des Disketteninhaltes der gerade im Laufwerk befindlichen Diskette. Die Anzeige kann jederzeit durch eine beliebige Taste angehalten und mit einem weiteren Tastendruck fortgesetzt werden. Durch <RUN/STOP> wird die Anzeige vorzeitig verlassen. Am Ende des Directorys führt jeder Tastendruck ins Menü zurück.

EXIT
Funktion: Verlassen des Programms, Neustart erfolgt über RUN.

NAME/ID
Der Name und die 5stellige ID der Diskette können geändert werden. Auf dem Bildschirm erscheint nun die Aufforderung »INSERT DISC«, es soll also die zu verändernde Diskette eingelegt werden. Ist dies geschehen, kann mit einem Tastendruck fortgefahren werden.

NAME
Funktion: Hiermit kann der Name einer Diskette ohne Datenverlust geändert werden. Hinter »OLD NAME« erscheint der bisherige Name der Diskette, wobei Steuercodes im Quote Mode dargestellt werden. Die Codes für <RETURN> und <SHIFT/RETURN> treten als Linkspfeil (wie man es von Textverarbeitungen kennt), <DEL> und <INST&DEL> als reverse »T« auf.

Unter der Bemerkung »NEW NAME« kann nun ein neuer Disketten-Name eingegeben werden, wobei alle Steuerzeichen außer <RETURN>, <SHIFT/RETURN>, <INST> und <DEL> übernommen werden können, falls vorher kein Anführungszeichen eingegeben wurde.
Die Bestätigung findet durch <RETURN> statt. Bei leerem Eingabefeld wird der alte Name übernommen. Die maximale Namenslänge beträgt 16 Zeichen, Überschreitungen werden einfach »abgekürzt«.
Funktionsweise: Ändern der Bytes 144–161 auf Block 18/0.

ID
Funktion: Ändern der ID einer Diskette ohne Formatierung. Für die Anzeige gelten dieselben Bedingungen wie unter »NAME« angegeben. Die maximale ID-Länge beträgt 5 Zeichen; auch hier wird die alte ID bei einem leeren Eingabefeld übernommen. <RETURN> dient wiederum als Bestätigung.
Funktionsweise: Ändern der Bytes 162–166 auf Block 18/0.

LOCK
Funktion: Schutz einer Diskette vor unbeabsichtigtem Löschen, Formatieren ohne ID-Angabe oder Veränderung durch BLOCK-WRITE-Befehle.
Funktionsweise: Änderung des dritten Bytes auf Block 18/0.

UNLOCK
Funktion: Rückgängigmachung von »LOCK«.

MENUE
Funktion: Rückkehr ins Hauptmenü

COMMAND
Funktion: Senden eines Floppy-Befehls ohne umständliche OPEN- und CLOSE-Befehle.
Beispiel: »R:NEU=ALT«
Funktionsweise: Senden des Kommandos über den Befehlskanal.

DEFORMAT
Funktion: Wiederherstellung eines Directorys, nachdem ohne ID formatiert wurde (Beispiel: OPEN 1,8,15,"N:LEER", nicht aber: OPEN 1,8,15,"N:LEER,LR"!).
Hinweis: Zuerst muß die Mindest-Block-Anzahl eingegeben werden (1–255), ab der das File in das Directory eingetragen wird. Bei nur einem Block ist ein Fehleintrag möglich, da es keinen weiteren Zeiger auf diesen Block gibt. Wird nur <RETURN> gedrückt, so erfolgt ein Rücksprung in das Hauptmenü.
Im folgenden werden nun alle Blockzeiger eingelesen (Anzeige: »READING POINTERS«), worauf sie analysiert werden (»ANALYZING«) und das neue Directory auf Diskette generiert

wird (»CREATING DIRECTORY«). Zuletzt erfolgt ein »VALIDATE« der Diskette, um die Programmblöcke in der BAM als belegt zu kennzeichnen und den verbleibenden Speicherplatz zu bestimmen.
Funktionsweise: Alle Disketten-Blöcke, auf die keine Sektorverkettung weist, werden als potentielle Anfangsblöcke von Files betrachtet. Der alte Name des Programms kann dabei nicht mehr restauriert werden, es erscheinen daher die Filenamen »1« bis »144«. Dabei empfiehlt es sich, alle wiederhergestellten Programme zu laden und ihnen dann ihre originalen Namen zurückzugeben bzw. nicht lauffähige Programme, die Abfallprodukte des DEFORMAT, zu löschen.

MANIPULATE
Funktion: Dient zur Veränderung der File-Parameter im Directory hinsichtlich Länge, Filetyp, Name etc.
Direkt nach der Anwahl wird das Directory eingelesen (»READING DIRECTORY«). Die Anwahl der zu verändernden Files geschieht durch <F5> und <F7> zur Auf- und Abwärtsbewegung. Die Parameter erscheinen jeweils im rechten oberen Anzeigefeld. Als Hilfe sind Pfeile auf die einzelnen Parameter gerichtet, an deren Ende stichwortartig die Bedeutung erklärt wird:

TRK/SE	Spur und Sektor des ersten Blockes
TYPE	Programmart
SEQ	sequentielle Datei
REL	relative Datei
PRG	Programmdatei
USR	User-Datei
DEL ???	gelöscht, aber nicht »gescratcht« illegaler Filetyp
–	endgültig gelöschtes File (erscheint nicht im normalen Directory)
LOCKED	Scratch-Schutz auf einem einzelnen File (im Directory erscheint »<«, wenn vorhanden)
OPEN	noch offenes File (»*«)
NAME	Filename
LENGTH	Dateilänge in Blöcken

NAME
Funktion: Änderung des Filenamens. Auch Steuerzeichen sind erlaubt, soweit sie nicht der Eingabesteuerung dienen:

<RETURN>	Bestätigung
<SHIFT/RETURN>	Bestätigung
<DEL>	Löschen des Eingabefeldes

Bei leerem Eingabefeld bleibt der alte Name erhalten.
Hinweis: Der Text »,8« oder »,8,1« läßt sich dadurch anhängen, daß man zum Beispiel erst »PROGRAMM«, dann <SHIFT/SPACE> und den Text »,8,1« eingibt. Das Ergebnis beim Einlesen des Directorys sähe dann wie folgt aus:

```
100 " PROGRAMM" ,8,1 PRG
```

Auch Farbsteuerzeichen und andere Codes zur »Verschönerung« bieten sich an.

TYPE/RECOVER
Funktion: Festlegung eines neuen Filetyps oder Wiederherstellen eines gelöschten Files. Die Anwahl der diversen File-Typen geschieht durch Buchstaben: <S> = SEQ, <P> = PRG, <D> = <DEL>, <U> = USR, <R> = REL, <?> = unbekannter Filetyp.
Da beim Löschen eines Files nur die Typkennung eines Programmes gelöscht wird und die Blöcke in der BAM (Spur 18, Sektor 0) als frei gekennzeichnet werden, muß oftmals nur der Filetyp neu gesetzt und die BAM auf den neuesten Stand gebracht werden. Nach der Wiederherstellung eines Files sollte also unbedingt ein VALIDATE erfolgen!
Das Ganze funktioniert allerdings nur erfolgreich, wenn nach dem Scratchen kein neues Programm auf die Diskette übertragen wurde, da sonst die Blöcke des gescratchten Files verloren sind.

LENGTH
Funktion: Veränderung des Längeneintrags durch Eingabe von maximal fünf Ziffern.

TRACK beziehungsweise **SECTOR**
Funktion: Änderung von Spur bzw. Sektor des ersten Blockes eines Programms.

CLOSE
Funktion: Schließen noch offener Files, um z.B. nach Fehlern während der Speicherung eines Files alle Daten zu retten. Nicht ordnungsgemäß geschlossene Files sind im Directory am Sternchen (z.B. »*SEQ«) zu erkennen.

LOCK beziehungsweise **UNLOCK**
Funktion: Herstellen/Löschen eines Scratch-Schutzes für einzelne Files. Dieser Schutz bewirkt optisch, daß bei einer Datei im Directory das Symbol »<« erscheint. Ein Schutz vor SAVE"<@>:NAME",8 ist dies jedoch nicht.

SCRATCH
Funktion: Löschen (Scratchen) einzelner Files.
Danach sollte ein VALIDATE erfolgen, da der Disc-Wizard dieses im Gegensatz zum Floppy-DOS nicht durchführt. Mit TYPE/RECOVER stellt man ein versehentlich gelöschtes Programm wieder her.

WRITE
Funktion: Schreiben des modifizierten Directory
Vor Anwahl dieses Punktes werden keine Veränderungen auf Diskette gesichert.

READ
Funktion: Einlesen eines neu zu bearbeitenden Directorys.
Bei Fehleingaben kann somit der ursprüngliche Inhalt eingelesen werden, aber auch Diskettenwechsel sind dadurch möglich.

MENUE
Funktion: Rücksprung ins Hauptmenü. Veränderungen am Directory werden nicht automatisch gespeichert.

DIR-SORTER
Funktion: Sortieren, Einfügen und Löschen von Files im Directory.
Hinweis: Direkt nach der Anwahl wird das Directory der im Laufwerk befindlichen Diskette eingelesen. Im Anschluß werden alle gelöschten Files aus dem Directory entfernt und sind auch nicht mit MANIPULATE wiederzuholen, wenn das bearbeitete Directory geschrieben worden ist, sondern nur durch DEFORMAT. Die Cursor- und Auswahlsteuerung geschieht wie bei MANIPULATE beschrieben.

INSERT
Funktion: Einfügen eines Trennstriches inmitten der File-Einträge, um die Übersichtlichkeit zu erhöhen (siehe Masterdiskette!).

POSITION
Funktion: Änderung der Reihenfolge der Dateien im Directory.
Mit <F5> und <F7> wird eine neue Position im Directory gesucht.

DELETE
Funktion: Vollständiges Löschen eines Eintrags aus dem Directory.
Danach sollte VALIDATE durchgeführt werden.

DEFINE LINE
Funktion: Umdefinieren des Trennstriches

READ beziehungsweise **WRITE**
Funktion: Lesen bzw. Schreiben des Directorys.

MENUE
Funktion: Rücksprung ins Hauptmenü.

MONITOR
Veränderung/Analyse eines Blockinhaltes.
Alle Eingaben erfolgen im Hexadezimalsystem, die Befehle werden im folgenden beschrieben.

INPUT
Funktion: Einlesen eines Blockes zur Bearbeitung.
Syntax: I spur sektor

OUTPUT
Funktion: Schreiben des Blockes.
Syntax: O spur sektor

FILL
Funktion: Füllen des Blocks mit einem beliebigen Wert.
Syntax: F byte

MEMORY DUMP
Funktion: Anzeige des Blockinhaltes
Syntax: M adresse
Fehlt »adresse«, so wird der gesamte Speicher angezeigt. Die Anzeige kann mit <CTRL>, <C=> oder <SHIFT> angehalten und mit <RUN/STOP> beendet werden. Änderungen nur im Direktmodus durch Cursorbewegungen und Eingaben.

EXIT
Funktion: Rücksprung ins Hauptmenü
Syntax: X

HELP
Funktion: Ausgabe aller Befehle in Hilfsmenü
Syntax: H

RESET
Funktion: Rückgängigmachung aller Veränderungen
Syntax: S

EDITED BLOCK
Funktion: Anzeige von Spur und Sektor des aktuellen Blockes
Syntax: B

STATUS
Funktion: Auslesen des Floppy-Fehlerkanals
Syntax: <@>

LAST BLOCK
Funktion: Einlesen des letzten Blockes vor Bearbeitung des aktuellen Blockes.
Syntax: L

NEXT BLOCK
Funktion: Einlesen des nächsten Blockes (laut Sektorverkettung)
Syntax: N

TEXT
Funktion: Eingabe eines Textes
Syntax: T adresse »TEXT«
Der Parameter »adresse« bedeutet, ab welchem Byte der Text einzufügen ist. Texte, die über das Blockende reichen, werden gekürzt.

ROTATE
Funktion: Zyklisches Linksrotieren der Bits
Syntax: R anzahl
Hinweis: »anzahl« ist ein Wert zwischen 00 und 07. Die Anwendung liegt in der (De-)Codierung von Texten oder Tabellen auf der Diskette; im Zusammenhang mit FIND TEXT lassen sich hiermit gefundene Texte decodieren und abändern.
Funktionsweise im Beispiel: Die Binärzahl 10101100 wird nach einmaliger Linksrotation zu 01011001, nach zweimaliger zu 10110010.
Wie man sehen kann, verschieben sich die Bits aller Datenbytes jeweils um eine Stelle nach links, wobei das linke Bit sich ja nicht weiter nach links verschieben läßt. Deshalb wird es auf der rechten Seite wieder angehängt. Da bei dieser Methode kein Bit verlorengeht, kann man damit Daten und Texte verschlüsseln.
Wie man hierbei sieht, ist nach der achten Rotation der Ursprungszustand wiederhergestellt, weshalb sich nur Rotationen von 00 bis 07 eignen. Bei einer 08-Rotation, also eigentlich 00-Rotation, bleiben Bit-Reihenfolge und Wert unverändert.

EOR
Funktion: Verknüpfung aller Bytes eines Blockes mit EOR.
Syntax: E wert
Dabei darf »wert« im Bereich von 00 bis FF liegen. Es dient zur (De-)Codierung von Daten.

HEX-DEC
Funktion: Umrechung einer Hexadezimalzahl in einen Dezimalwert
Syntax: $zahl
Dabei ist »zahl« eine zwei- oder vierstellige Hexadezimalzahl.

DEC-HEX
Funktion: Umrechung einer Dezimal- in eine Hexadezimalzahl
Syntax: #zahl
Hinweis: Die maximale Zahl ist 65535.

PRINT
Funktion: Ausgabe des Blockinhalts auf Drucker mit Geräteadresse 4
Syntax: P

CATALOG
Funktion: Ausgabe des Disketteninhaltes
Syntax: C

DISC COMMAND
Funktion: Senden eines Diskettenbefehls an die Floppy
Syntax: *befehl
Hinweis: Mit »befehl« ist ein Befehlstext gemeint.

FIND TEXT
Funktion: Suchen nach eventuell verschlüsselten Texten auf der Diskette.
Hinweis: Bei Auffinden eines Textes erscheinen die Parameter als EOR-Wert, ROTATE-Wert und Spur/Sektor-Angabe.
Nach Drücken der Leertaste wird weitergesucht, mit jeder anderen Taste kehrt man ins Hauptmenü zurück. Beim Suchen werden immer zwei Blöcke gleichzeitig eingelesen, um auch sektorübergreifende Texte zu finden.

1 WATCH TRACK(S)
Suchen nach Texten auf ganzen Spuren

1.1 FIND TEXT
Eingabe des Suchtextes

1.2 START TRACK
Eingabe der ersten Spur, ab welcher gesucht wird

1.3 END TRACK
Eingabe der letzten Spur, bis zu welcher gesucht wird

1.4 EOR-CODE

Eingabe des EOR-Wertes für die Decodierfunktion. Bei einem Wert gleich 0 wird nach unverschlüsselten Texten gesucht.

1.5 ROTATE-CODE

Eingabe der Anzahl, wie oft die Bits rotiert werden sollen. Bei der Eingabe sind Werte von 00 bis 07 zugelassen, bei einem Rotationswert von 0 wird nach unverschlüsselten Texten gesucht.

1.6 EOR-ROTATE

Reihenfolge der Decodierung: erst EOR und dann ROTATE, oder umgekehrt; Einstellung durch »y« für »zuerst EOR«, »n« für umgekehrte Reihenfolge, mit Hilfe der Cursortasten.

1.7 CONTINUOUSLY

– Anwendung von Punkt 1.4 bis 1.6 in allen Kombinationen – zur Beschleunigung der Vergleiche (2 Millionen Vergleiche pro Block) werden Interrupt und Bildschirm abgeschaltet. Zur Kontrolle werden aber in einem bestimmten Rhythmus die Bildschirmfarben verändert.

Die Dauer für einen Block schwankt zwischen 8 und 10 Minuten. Bei der Endabfrage »ARE YOU SURE« kann wiederum mit den Cursortasten zwischen »YES« und »NO« entschieden werden, worauf <RETURN> als Bestätigung dient.

2 FOLLOW POINTERS

Blockverfolgung entsprechend der Sektorverkettung

2.1 FIND TEXT bis 2.2 START TRACK

wie 1.1/1.2

2.2 START SECTOR

Eingabe des Startsektors

2.3 EOR-CODE bis 2.7 CONTINUOUSLY

wie 1.4 bis 1.7

3 WATCH TWO SECTORS

Suche nach Text in nur zwei zusammenhängenden Blöcken

4 MENUE

Rücksprung ins Hauptmenü

11.7 ProDisc – eine leistungsfähige Diskettenverwaltung

Nicht schon wieder eine Diskettenverwaltung, werden Sie vielleicht denken. Doch wird die Anwendung Ihre Meinung garantiert ändern. Etwa durch die komfortable Bedienung, die durch eine »Benutzeroberfläche« realisiert wurde. Dies bedeutet, daß keine umständlichen Befehle eingegeben werden müssen, sondern Sie nur noch auf ein »Icon« (Bildsymbol) deuten, wie Sie es von GEOS kennen.
Ganz zu schweigen von der Datenmenge, die sich damit erfassen läßt: 1745 Programmtitel auf einmal dürfen im Speicher stehen. Weitere Pluspunkte: Sortieren nach diversen Kriterien, unterschiedliche Delete-Modi, Mehrfacheditierung, anpaßbare Druckroutine und gute Übersichtlichkeit durch Window-Technik. Alles in allem ein sehr brauchbares Werkzeug für alle, die sich bei ihren Disketten nicht mehr zurechtfinden.

Laden und Starten

Nach LOAD "PRODISC",8,1 und RUN erscheint kurze Zeit später ein Titelbild; haben Sie dieses lange genug bewundert, so drücken Sie die Leertaste. Wieder eine Zeit später, wird der ProDisc-Arbeitsbildschirm aufgebaut. Es erscheint das immer sichtbare Hauptmenü mit der am rechten Rand befindlichen Symbol-Leiste. Des weiteren ist am oberen Bildschirmrand noch ein Statusfeld zu sehen, worauf wir noch zu sprechen kommen, sowie ein blinkender Auswahl-Pfeil.

Die Symbol-Leiste

Die rechts befindlichen Bildsymbole, die mit den Cursortasten angewählt und durch Drücken von <RETURN> aktiviert werden, haben folgende Bedeutung (von oben nach unten):

- `Icon 1 (Diskettenzugriff)`

  Hier können Sie Datensätze laden, speichern und mit dem Speicherinhalt vergleichen (verifizieren). Die Datei wird automatisch unter dem Namen »CCT-DATA-FI« gespeichert, ein bestehender Eintrag wird überschrieben. Fehler im Bus-Betrieb, an den Benutzer gerichtete Aufforderungen oder Anfragen zeigt das Programm automatisch in der Kommentarzeile (...) im Statusfeld an. Ins Hauptmenü kehrt man mit <←> zurück.

- `Icon 2 (Löschen)`

  Das Löschen erfolgt über drei Kriterien (Alphabet, Seite und ID). Dabei ist der Einsatz des Pfundsymbols als Joker möglich. Um zum Beispiel alle Programmnamen zu löschen, die mit »SI« beginnen, gibt man nach Drücken der Alphabet-Taste <1> folgendes ein: SI<£>, gefolgt von <RETURN>. Dies ist der INCLUDE-Modus. Zusätzlich können Sie auch den EXCLUDE-Modus wählen (s. Icon 7); dann würden alle Programme außer denen, die mit »SI« beginnen, gelöscht.

- `Icon 3 (Listen der Datei)`

  Es gelten dieselben Kriterien wie im Delete-Modus (Icon 2). Um die gesamte Datei zu listen, drückt man einfach die Alphabet-Taste <1>, gefolgt von <RETURN>. Durch Drücken von <SPACE> wird der nächste Programm-Name gelistet. Soll ein Name verändert werden, so gibt man statt <SPACE> einfach <E> wie Editieren ein. Nun können Sie sämtliche Angaben ändern. Die Windows verläßt man jederzeit durch <←>.

- `Icon 4 (Diskette aufnehmen)`

  Von der eingelegten Diskette werden zunächst Name und ID gelesen und angezeigt. Diese Daten sind durch Drücken von <RETURN> editierbar; die geänderten Daten schreibt das Programm anschließend auf die Diskette zurück und startet einen neuen Lesevorgang.
  Wollen Sie den Header nicht ändern, so fragt Sie das Programm nach Eingabe von <N> (Nein) nach der Diskettenseite. Hier geben Sie bitte <A> für die Seite, auf der sich das Disketten-Etikett befindet, und <B> für die Rückseite ein. Nach der Seitenwahl werden nun die Programmtitel gelistet. Durch die Cursortasten lassen sich die Einträge seitenweise bis zum Ende des Directorys listen. Mit <CLR/HOME> gelangt man wieder zum Anfang des aktuellen Windows. Durch <RETURN> wird der Eintrag aufgenommen, auf schon bestehende Einträge weist die Kommentarzeile hin.
  In der Statuszeile lassen sich nun ID, die Seite, die Zahl der auf der Diskette enthaltenen Einträge und die Anzahl der noch freien Einträge ablesen. Auch hier besteht die Möglichkeit, mit der Taste <E> die Programm-Namen noch vor der Aufnahme zu editieren.
  Ein Hinweis am Rande: Gelegentlich tritt der Fall auf, daß Sie mit der Cursor-Auswahl aus dem Bildschirm hinaus und durch den Speicher fahren können. Probieren Sie das lieber nicht aus, sondern fahren Sie mit dem Cursor wieder nach oben. Es könnte sonst ein Teil der Datei zerstört werden. Für den seltenen Fall, daß ProDisc einmal »abstürzt«, so geben Sie einfach LOAD"PRORETTER",8,1 ein, und Sie befinden sich kurze Zeit später im Hauptmenü von ProDisc, wo Sie nun ungehindert weiterarbeiten können.

- `Icon 5 (Sortieren)`

  Zur Auswahl stehen drei Sortierkriterien: Alphabet, Diskettenseiten und ID. Wartezeiten sind nicht auszuschließen. Zusätzlich ist es möglich, über Autosort (siehe Icon 7) die Einträge automatisch nach Aufnahme alphabetisch sortieren zu lassen, was natürlich viel schneller vonstatten geht.

- `Icon 6 (Drucken)`

  Es wird die komplette Datei im aktuellen Zustand ausgedruckt. Wahlweise geschieht dies mit oder ohne CBM-Zeichensatz und Einzelblatt- beziehungsweise Endlosausdruck (siehe Icon 7).

- `Icon 7 (Optionen)`

  Verlassen Sie damit das ProDisc-System oder ändern Sie die wichtigsten Parameter. Durch Drücken der Anfangsbuchstaben der Optionen werden die jeweiligen Flags gesetzt oder gelöscht (ausgefüllte Kreisfläche bedeutet gesetzt, Kringel gelöscht).

  CBMZ.SATZ: Ausdruck mit CBM-Zeichensatz für MPS-Drucker oder normalem Zeichensatz für Fremddrucker (Epson,..).

  EZ-BLATT: Das Programm wartet entweder bei jedem fertigen Blatt, bis ein neues eingelegt wird (ausgefüllte Kreisfläche) oder druckt ohne Unterbrechung (Kringel).

  INCLUDE: Löschen nach bestimmten Kriterien (s. Icon 2), normalerweise INCLUDE.

  AUTOSORT: Ist diese Funktion aktiv, sortiert das Programm neue Einträge gleich bei der Übernahme ein.

- Allgemein gilt: Die jeweiligen Unterprogramme und Windows können jederzeit durch < ← > verlassen werden. Die Taste <RESTORE> darf man jedoch nicht betätigen, weil das Programm sonst abstürzt und PRORETTER zu laden wäre.

11.8 Disketten-Reparatur mit Reformat

Haben Sie auch schon einmal versehentlich eine Diskette neu formatiert? Wenn ja, dann brauchen Sie »Reformat«. Denn mit diesem Programm läßt sich die Diskette sehr einfach reparieren. Denn wenn eine Diskette ohne Angabe einer ID formatiert wurde, präsentiert sich ein leeres Directory mit 664 freien Blöcken. Doch dieser Zustand trügt: In Wirklichkeit sind noch alle Daten auf der Diskette vorhanden, nur der Zugang dazu ist verbaut.
Es werden bei der Formatierung ohne ID nur die BAM (Block 18/0) und der erste Sektor des Directorys (18/1) formatiert, weshalb dieser Vorgang auch nur wenige Sekunden dauert. Da in jedem Sektor des Directorys acht Files verwaltet werden, sind außer den ersten acht Filenamen alle weiteren Filenamen noch vorhanden.
Um die Programme auf der Diskette zu retten, muß man »nur« die gelöschten Sektoren 18/0 und 18/1 rekonstruieren. Man benötigt dazu vor allem die Start-Blöcke der Programme. Um die Start-Blöcke zu finden, müssen die ersten Bytes aller Blöcke von der Diskette eingelesen werden, da diese die Zeiger auf die jeweils folgenden Blöcke enthalten. Durch diese Zuordnung kristallisieren sich bald die Programmanfänge heraus. »Reformat« erledigt dies (Listing 11.2).

Listing 11.2: REFORMAT

```
1 REM ********************************
2 REM *        REFORMAT V4.0         *
3 REM *                              *
4 REM *  (C) 1985 BY GEORG BURGER    *
5 REM *     ROIDERSTRASSE 18         *
6 REM *       8051 ZOLLING           *
7 REM ********************************
8

9

10 POKE53280,0
   POKE53281,0
   POKE646,5
   PRINTCHR$(142)
20 OPEN1,8,15,"I"
   CLOSE1
   GOTO1000
97

98 REM  ---------ROUTINEN-------------
99

100 CLOSE5
    OPEN5,8,5,"#2"
    RETURN
200 INPUT#15,Y1$,Y2$,Y3$,Y4$
210 IFVAL(Y1$)=0THENRETURN
220 PRINT"(DOWN)"Y1$" "Y2$" "Y3$" "Y4$
230 PRINT"(DOWN)   USE (RVS) ← (OFF) TO EXIT OR (RVS) F7 (OFF) TO CONTINUE(DOWN)
"
240 POKE198,0
250 GETA$
    IFA$=""THEN250
260 IFA$="←"THENRUN
270 IFASC(A$)=136THENRETURN
280 GOTO250
300 PRINT"(CLR)________________________________________";
310 PRINT"(RVS)   REFORMAT  V4.0    BY  GEORG BURGER   (OFF)";
320 FORI=1TO7
    PRINT"|                                     |";
    NEXTI
330 PRINT"______________________________________"
    RETURN
400 PRINT"OLD DIRECTORY-REGISTER
"
    PRINT
410 PRINT"NAME",DN$(D1)
420 PRINT"BLOCKS",DL(D1)
430 PRINT"TRACK",DS(D1)
440 PRINT"SECTOR",DB(D1)
450 EF$="Y"
    RETURN
500 PRINT"(DOWN)                 (RVS) PRESS ANY KEY (OFF)"
    POKE198,0
510 GETA$
    IFA$=""THEN510
520 RETURN
600 D1=0
610 IF(DS(D1)=AS)AND(DB(D1)=AB)THEN400
620 D1=D1+1
    IFD1>=DPTHENRETURN
630 GOTO610
700 MB=NB
```

```
    NB=NB+3
710 IFNB=19THENNB=2
720 IFNB=20THENNB=3
730 RETURN
997

998 REM --------- MENUE ---------------
999

1000 OPEN15,8,15
     OPEN5,8,5,"#2"
1010 DIMS(35,21),B(35,21),BL(35),DI$(29),DI(29),A$(255)
1020 DIMSV(35,21),BV(35,21),US(35,21),DS(144),DB(144),DL(144),DN$(144),DT(144)
1030 FORI=1TO17
     BL(I)=20
     NEXT
     FORI=18TO24
     BL(I)=18
     NEXT
1040 FORI=25TO30
     BL(I)=17
     NEXT
     FORI=31TO35
     BL(I)=16
     NEXT
1050 GOSUB300
     POKE214,4
     POKE211,0
     SYS58640
1060 PRINT"(RGHT)    (RVS) F1 (OFF)   RECONSTRUCT DIRECTORY
1070 PRINT"(RGHT)    (RVS) F7 (OFF)   EXAMINE DIRECTORY"
     POKE198,0
1080 GETA$
     IFA$=""THEN1080
1090 IFASC(A$)=133THENGOSUB100
     GOTO2000
1100 IFASC(A$)=136THENGOSUB100
     GOSUB4000
     GOTO1050
1110 GOTO1080
1997

1998 REM --- RECONSTRUCT DIRECTORY ----
1999

2000 GOSUB300
     POKE214,4
     POKE211,0
     SYS58640
2010 PRINT"(RGHT)          RECONSTRUCT DIRECTORY
2020 PRINT"(RGHT)             USE (RVS) ← (OFF) TO EXIT
2030 PRINT"(DOWN)(DOWN)________________________________________";
2040 PRINT"(RVS) TRACK   SECTOR   TO   TRACK    SECTOR (OFF)"
2050 FORS=1TO35
     FORB=0TOBL(S)
2060 PRINT#15,"B-R
     "5;0;S;B
     GOSUB200
2070 PRINT#15,"B-P
     "5;0
     GOSUB200
2080 GET#5,E$
2090 GET#5,F$
2100 E=ASC(E$+CHR$(0))
2110 F=ASC(F$+CHR$(0))
2120 PRINT"  "S,B,"    "E,"    "F
```

```
2130 GETA$
     IFA$=""THEN2150
2140 IFA$="←"THENCLOSE15
     RUN
2150 IFE=75ANDF=1THENUS(S,B)=-2
     GOTO2200
2160 IFE>35THENE=0
2170 IFF>21THENF=21
2180 S(S,B)=E
     B(S,B)=F
2190 SV(E,F)=S
     BV(E,F)=B
     US(E,F)=US(E,F)+1
2200 NEXT
     NEXT
2210 REM ------ CRITICAL BLOCKS -------
2220 GOSUB300
     POKE214,5
     POKE211,0
     SYS58640
2230 PRINT"(RGHT)              CRITICAL BLOCKS
2240 PRINT"(DOWN)(DOWN)(DOWN)(DOWN)(RVS) TRACK    SECTOR   LENGHT (OFF)"
     PRINT
2250 FORS=1TO35
2260 FORB=0TOBL(S)
2270 IFUS(S,B)<2THEN2290
2280 PRINT"   "S,B,US(S,B)
2290 NEXT
     NEXT
2300 PRINT"(DOWN)          (RVS) END OF CRITICAL BLOCKS (OFF)"
2310 GOSUB500
     GOSUB300
     POKE214,4
     POKE211,0
     SYS58640
2320 PRINT"(RGHT)   USE OLD DIRECTORY TO NAME FILES ?
2330 PRINT"(RGHT)                   (Y/N)
2340 GETDA$
     IFDA$=""THEN2340
2350 IFDA$<>"N"THENDA$="Y"
     GOSUB4030
2360 FORS=1TO35
2370 IFS=18THEN2550
2380 FORB=0TOBL(S)
2390 S1=S
     B1=B
2400 IF(S=1)AND(S(S,B)=0)AND(B(S,B)=1)THEN2540
2410 IFUS(S1,B1)<0THEN2540
2420 IFUS(S1,B1)>0THENS2=SV(S1,B1)
     B2=BV(S1,B1)
     S1=S2
     B1=B2
     GOTO2420
2430 PRINT"(CLR)—————————————(RVS) FILE-START (OFF)—————————————";
     BA=1
     AS=S1
     AB=B1
2440 PRINTS1,B1
     US(S1,B1)=-1
2450 S2=S(S1,B1)
     B2=B(S1,B1)
     S1=S2
     B1=B2
2460 IFS1<>0THENBA=BA+1
     GOTO2440
```

```
2470 PRINT"————————————(RVS) FILE-END (OFF)————————————(DOWN)"
2475 PRINT"LENGHT
     ";BA;"BLOCKS"
     PRINT
2480 IFBA<2THEN2540
2490 IFDA$="Y"THENEF$="N"
     GOSUB600
2500 PRINT"(DOWN)WRITE BACK IN DIRECTORY (Y/N)"
     POKE198,0
     PRINT
2510 GETE$
     IFE$=""THEN2510
2520 IFE$="Y"THEN2570
2530 IFE$<>"N"THEN2510
2540 NEXTB
2550 NEXTS
2560 GOTO1050
2570 REM -------- WRITE BAM -----------
2580 S1=AS
     B1=AB
2590 PRINT#15,"B-A
     "0;S1;B1
     GOSUB200
2600 S2=S(S1,B1)
     B2=B(S1,B1)
     S1=S2
     B1=B2
2610 IFS1<>0THEN2590
2620 REM ----- WRITE DIRECTORY --------
2630 MB=1
     NB=4
2640 PRINT#15,"U1
     "5;0;18;MB
     GOSUB200
2650 PRINT#15,"B-P
     "5;0
     GOSUB200
2660 GET#5,NS$,NB$
2670 NS=ASC(NS$+CHR$(0))
2680 IFNS<>0THENGOSUB700
     GOTO2640
2690 C=0
2700 GET#5,A$,B$,C$
2710 IFA$=""THENA$="0"
2720 IFB$=""THENB$="0"
2730 IFC$=""THENC$="0"
2740 IFA$="0"THENIFB$="0"THENIFC$="0"THEN2910
2750 FORI=1TO27
     GET#5,A$
     NEXT
2760 C=C+1
     IFC<8THENGET#5,A$,A$
     GOTO2700
2770 IFMB<18THEN2810
2780 PRINT"DIRECTORY IS FULL !"
2790 GETA$
     IFA$=""THEN2790
2800 GOTO1050
2810 PRINT#15,"B-P
     "5;0
     GOSUB200
2820 PRINT#5,CHR$(18);
2830 PRINT#5,CHR$(NB);
2840 PRINT#15,"U2
     "5;0;18;MB
```

```
     GOSUB200
2850 GOSUB700
2860 PRINT#15,"U1
     "5;0;18;MB
     GOSUB200
2870 PRINT#15,"B-P
     "5;0
     GOSUB200
2880 PRINT#5,CHR$(0);CHR$(255);
     FORI=0TO253
     PRINT#5,CHR$(0);
     NEXT
2890 PRINT#15,"U2
     "5;0;18;MB
     GOSUB200
2900 GOTO2640
2910 PRINT"TRACK 18 SECTOR"MB"POSITION"C+1
     PRINT
2920 P=2+C*32
2930 PRINT#15,"B-P
     "5;P
     GOSUB200
2940 DI$(0)=CHR$(130)
2950 DI$(1)=CHR$(AS)
2960 DI$(2)=CHR$(AB)
2970 FORI=3TO18
     DI$(I)=CHR$(160)
     NEXT
2980 IFDA$="Y"ANDEF$="Y"THENPRINT"        "DN$(D1)"(UP)"
2990 INPUT"NAME ";N$
3000 N$=LEFT$(N$,16)
3010 FORI=0TOLEN(N$)-1
3020 DI$(3+I)=MID$(N$,I+1,1)
3030 NEXT
3040 FORI=19TO27
     DI$(I)=CHR$(0)
     NEXT
3050 BH=INT(BA/256)
     BL=BA-256*BH
3060 DI$(28)=CHR$(BL)
3070 DI$(29)=CHR$(BH)
3080 FORI=0TO29
3090 PRINT#5,DI$(I);
3100 NEXT
3110 PRINT#15,"U2
     "5;0;18;MB
     GOSUB200
3120 GOTO2540
3997

3998 REM ---- EXAMINE DIRECTORY -------
3999

4000 GOSUB300
     POKE214,4
     POKE211,0
     SYS58640
4010 PRINT"(RGHT)             EXAMINE DIRECTORY
4020 PRINT"(RGHT)              USE (RVS) ← (OFF) TO EXIT
4030 PRINT
4040 MB=1
     DP=0
4050 PRINT#15,"U1
     "5;0;18;MB
     GOSUB200
4060 PRINT#15,"B-P
```

```
     "5;0
     GOSUB200
4070 GET#5,NS$,NB$
4080 NB=ASC(NB$+CHR$(0))
4090 NS=ASC(NS$+CHR$(0))
4100 PRINT"(DOWN)________________________________________";
4110 PRINT"      TRACK 18  SECTOR"MB"-->"NS" "NB
4120 PRINT"________________________________________";
4130 GETA$
     IFA$=""THEN4150
4140 IFA$="←"THENCLOSE15
     RUN
4150 IFNS=75ANDNB=1THEN4440
4160 PRINT
4170 PRINT"(RVS)BLOCKS"TAB(7)"NAME"TAB(26)"TYP"TAB(31)"TRA."TAB(36)"SEC.(OFF)"
4180 FORC=0TO7
4190 GETA$
     IFA$=""THEN4210
4200 IFA$="←"THENCLOSE15
     RUN
4210 FORI=0TO29
4220 GET#5,DI$(I)
4230 NEXTI
4240 IFC<>7THENGET#5,A$,A$
4250 FORI=0TO29
     DI(I)=ASC(DI$(I)+CHR$(0))
     NEXT
4260 KW$=" NO MORE FILES IN SECTOR"
4270 IFDI(0)=0ANDDI(1)=0ANDDI(2)=0THENPRINT"(DOWN)       (RVS)"KW$;MB"(LEFT) (OFF
)"
     PRINT
     GOTO4440
4280 DS(DP)=DI(1)
     DB(DP)=DI(2)
4290 DL(DP)=DI(29)*256+DI(28)
4300 DN$(DP)=""
     FORI=3TO18
     IFDI(I)<>160THENDN$(DP)=DN$(DP)+DI$(I)
4310 NEXTI
4320 DT(DP)=DI(0)
4330 DP=DP+1
4340 PRINTDI(29)*256+DI(28);TAB(7);CHR$(34);
4350 FORI=3TO18
     IFDI(I)<>160THENPRINTDI$(I);
4360 NEXT
     PRINTCHR$(34);
4370 BY=DI(0)AND(NOT128)
     IFBY=0THENPRINTTAB(26);"DEL";
4380 IFBY=1THENPRINTTAB(26);"SEQ";
4390 IFBY=2THENPRINTTAB(26);"PRG";
4400 IFBY=3THENPRINTTAB(26);"USR";
4410 IFBY=4THENPRINTTAB(26);"REL";
4420 PRINTTAB(30);DI(1);TAB(35);DI(2)
4430 NEXTC
4440 MB=MB+1
     IFMB<>19THEN4050
4450 IFDA$="Y"THENRETURN
4460 GOSUB500
     RETURN
```

Nach Starten dieses Basic-Programms mit RUN hat man die beschädigte Diskette ins Laufwerk einzulegen. Im Auswahlmenü untersucht man mit <F7> die Sektoren 1 bis 18 der Spur 18. Es werden in diesem Programmpunkt die untersuchten Blöcke mit den Zeigern auf die nächsten Blöcke aufgelistet. Zeigt ein Sektor auf 0/255, so ist dieser Sektor der letzte Sektor mit Einträgen. Die Reihenfolge der Sektoren ist dabei nicht linear (1,2,3,4...), sondern geht in Dreierschritten von 1 bis 18 (1,4,7,10,13,16;2,5,8,11,14,17;3,6,9...).
Zeigt ein Sektor auf 75,1, dann ist dieser Sektor nicht benutzt, also noch im frisch formatierten Zustand und hat deshalb keine Daten gespeichert. »NO MORE FILES IN SECTOR x« weist darauf hin, daß in diesem Sektor nicht alle acht Einträge vorhanden sind.
Hatte das Directory nicht mehr als acht Einträge, so wird man keinen Programmnamen finden; hatte es jedoch mehr als acht Einträge, so sind ab Sektor 4 (dem Folge-Sektor vom gelöschten Sektor 1) die restlichen Einträge mit Namen, Anzahl der Blöcke, Programmtyp und Startblock (Spur/Sektor) angegeben.
Diesen Programmpunkt verläßt man durch Drücken von < ← > vorzeitig. Am Schluß des Programmteils gelangt man durch einen Tastendruck wieder ins Hauptmenü. Mit <F1> wird die eigentliche Rekonstruktion des Directorys gestartet. Es werden zuerst die Block-Zeiger eingelesen. Als nächstes werden die kritischen Blöcke, also die Blöcke, auf die mehrere Blöcke zeigen, aufgelistet. Diese Blöcke sollte man sich notieren.
Nun muß man entscheiden, ob man die noch vorhandenen Programmnamen im Directory für die Rekonstruktion verwenden will. Wenn ja, dann wird zusätzlich das Directory auf noch vorhandene Namen untersucht. Im Anschluß werden die gefundenen Programme aufgelistet. Man hat nun zu entscheiden, welche Programme man wieder übernehmen möchte. Beantwortet man die Frage »WRITE BACK IN DIRECTORY« positiv, dann wird das Programm in die BAM eingetragen.
Hierbei können einige Probleme auftreten. Kritische Blöcke sollten nicht in das Directory zurückgeschrieben werden, da Blöcke, die als belegt gekennzeichnet werden sollten und schon belegt sind, die Fehlermeldung »NO BLOCK« auslösen. In diesem Fall kann man sich durch das im Programm vorgesehene Drücken der F7-Taste durch die kritischen Blöcke hangeln, bis ein normales Programm gefunden wird.
Wichtig ist, daß auf der zu reformatierenden Diskette 664 Blöcke frei sind, da sonst die gefundenen Programme nicht richtig in die BAM eingetragen werden können, was ebenfalls die Fehlermeldung »NO BLOCK« zur Folge hat.
Wenn im Directory noch der alte Name zu finden war, so wird dieser vom Programm vorgeschlagen; wenn der alte Name schon gelöscht war, ist ein neuer einzugeben.
Ist das gesuchte Programm eingetragen, läßt sich die Reformatierung ohne weiteres beenden. Programme mit nur einem Block werden ignoriert, da diese keine eindeutige Zuordnung zulassen.
Bei auftretenden Fehlern wie READ ERROR oder NO BLOCK kann man das Programm durch <F7> fortsetzen oder durch < ← > neu starten. Bei Fortsetzung mit <F7> muß man sich unter Umständen durch mehrmaliges Drücken von <F7> durch eine ganze Spur wühlen.

11.9 Schnelles Formatieren in 11 Sekunden

Das 1541-Floppy-Laufwerk gehört nicht nur beim Laden, sondern auch beim Formatieren einer Diskette zu den langsameren seiner Gattung. »Fast Format« ändert das.
Wenn man beispielsweise ein ganzes Paket (zehn Disketten) nacheinander formatieren möchte, sitzt man fast 15 Minuten vor dem Computer. Fast Format reduziert diesen Vorgang auf 11 Sekunden pro Diskette, bei zehn Disketten, auf nicht einmal drei Minuten. Das eigentliche Maschinenprogramm wird über

```
LOAD"FAST FORMAT",8,1
```

geladen und ist nach NEW betriebsbereit.
Die neue Formatierroutine wird durch folgenden Befehl aufgerufen:

```
SYS 49152,"name","ID",gerät
```

Etwas einfacher geht es mit einem kleinen Hilfsprogramm:

Listing 11.3: Hilfsprogramm zu FAST FORMAT

```
0 POKE 53280,0
  POKE 53281,0
  PRINT"(CLR)(GRN)"
1 PRINT SPC(12)"FASTFORMAT 11.0(DOWN)"
2 PRINT SPC(13)"VON JAN KUSCH(DOWN)"
10 INPUT"(DOWN)DISK-NAME   ";N$
20 INPUT"(DOWN)DISK-ID     ";I$
30 INPUT"(DOWN)LAUFWERK-NR.";D
35 IF D<>8 AND D<>9 THEN30
40 SYS49152,N$,I$,D
```

Eine kleine Einschränkung aber ist unvermeidbar: In der Regel gibt es keine Probleme mit schnell formatierten Disketten, aber bei älteren und verstellten Laufwerken sind Fehler möglich. In diesem Fall ist dann der normale Formatier-Befehl vorzuziehen.

11.10 Disk-Füller

Das Problem ist wohl jedem bekannt, der eine gut sortierte Programmsammlung sein eigen nennt: Alle thematisch gleichen Programme bis auf eines passen auf eine Diskettenseite. Und gerade dieses letzte Programm benötigt nur einige wenige Blöcke. Aber hier kann in Notfällen Abhilfe geschaffen werden. Auf der Spur 18, auf der sich das Directory befindet, sind meist noch einige Blöcke frei, die das DOS der 1541 nicht zur Verfügung stellt. Zu diesem Zweck

werden Blöcke von schon auf der Diskette befindlichen Programmen auf freie Blöcke der Spur 18 umkopiert und die Blockzeiger angepaßt. Die ursprünglichen Programmblöcke stehen dann zur freien Verfügung.

Das Programm ist weitestgehend selbsterklärend, hier aber einige wichtige Hinweise:

- Nach dem Laden darf das Programm erst mit RUN gestartet werden, wenn die zu behandelnde Diskette im Laufwerk liegt.
- Es dürfen nur Disketten mit einwandfreier BAM behandelt werden. Im Zweifelsfalle ist vorher Validate durchzuführen (OPEN 1,8,15,"V":CLOSE1).
- Das Programm arbeitet nur, wenn weniger als das gesamte Directory belegt ist. Die Anzahl der freigemachten Blöcke hängt von der Auslastung der Spur 18 ab. Bei wenigen Directory-Einträgen werden also mehr Blöcke gewonnen als bei vielen.

Bei der Vorderseite der Masterdiskette lag genau die andere Problemstellung vor: Wie konnte das Directory, das nicht mehr auf Spur 18 paßte, auf Kosten von eigentlich für Programme und Daten vorgesehenen Blöcken ausgelagert werden? Mehr dazu im Anhang.

Listing 11.4: DISK-FUELLER

```
100 rem  **************************
110 rem  *                        *
120 rem  *                        *
130 rem  *     disk   fueller     *
140 rem  *                        *
150 rem  *                        *
160 rem  *                        *
170 rem  *   von  uwe  gerlach    *
180 rem  *                        *
190 rem  *     bruehlstr. 23      *
200 rem  *                        *
210 rem  *                        *
220 rem  *     6440  bebra 1      *
230 rem  *                        *
240 rem  *                        *
250 rem  *                        *
260 rem  *                        *
270 rem  *     im april 1985      *
280 rem  *                        *
290 rem  *                        *
300 rem  **************************
400 :
500 :
1000 poke53280,0: poke53281,0: rem farbe
1010 print chr$(14);"(clr)(gry2)  VC 1541    D i s k   F u e l l e r":print:prin
t:print
1015 print"Dieses Programm wurde geschrieben von:":print:print
1020 print,"(lblu)  Uwe Gerlach": print,"  Bruehlstr. 23": print: print,"  6440
Bebra 1(gry2)"
1025 print:print:print"Es schafft auf vollen Disketten noch"
1030 print"etwas Platz, indem es die eigentlich"
1035 print"fuer das Directory vorgesehene Spur 18"
1040 print"belegt.":print:print
1045 :
1050 open 1,8,15,"i": rem  fuer befehle
1060 open 2,8,2,"#" : rem  pufferkanal
1090 :
```

```
1100 rem  ******************************
1110 rem    directorylaenge ermitteln
1120 rem  ******************************
1130 :
2000 dim t(25): dim s(25): dim e(25)
2010 print# 1, "u1:"2;0;18;0: rem  bam
2020 get# 2,t$,s$: rem  zeiger auf dir.
2030 t(0)=asc(t$+chr$(0)): rem  track
2040 s(0)=asc(s$+chr$(0)): rem  sector
2050 t=t(0): rem  track  dir. anfang
2060 s=s(0): rem  sector dir. anfang
2070 b=1: rem  blockzaehler
2100 print# 1, "u1:"2;0;t;s
2110 get# 2,t$,s$: rem  zeiger
2120 t=asc(t$+chr$(0))
2130 s=asc(s$+chr$(0))
2140 if t=0 or t>35 or s>20 or b=25 then 2400:  rem  letzter directory-block
2150 t(b)=t: s(b)=s: rem  zeiger merken
2155 e(b-1)=8: rem  eintraege im block
2160 b=b+1: rem  zaehler erhoehen
2190 goto 2100 : rem  naechster block
2400 m=b: rem  dir. blockzahl
2500 f=0: rem  zaehler freie eintraege
2510 for i=2 to 226 step 32
2520 : print# 1, "b-p:"2,i: rem zeiger
2530 : get# 2,w$: rem  filetyp
2540 : if w$="" then f=f+1: rem  frei
2590 next i: rem  alle eintr. im block
2600 e(b-1)=8-f: rem  eintraege/block
2610 if f=8 and m=1 then 7800: rem leer
2700 print "Genuegen";f;"freie Directory-Plaetze ? ";
2750 get a$: if a$="" then 2750
2760 if a$<>"n" then a$="j"
2770 print a$: print: print: print
2790 :
3000 rem  ******************************
3010 rem    leere dir. bloecke festst.
3020 rem  ******************************
3030 :
3050 print# 1, "u1:"2;0;18;0: rem  bam
3060 print# 1, "b-p:"2,73: rem  spur 18
3100 b=0 : rem  blockzaehler
3110 l=19: rem  zahl leere dir-bloecke
3120 dim z(25): rem  zustand der blocks
3130 for i=1 to 3: rem  bytes spur 18
3140 : get# 2, w$: rem  belegungscode
3150 : w=asc(w$+chr$(0))
3160 : for j=1 to 8: rem  dualstellen
3170 :  w=w/2
3180 :  if w=int(w) then z(b)=1
3190 :  if z(b)=0 and a$="n" then a$="": z(b)=1: rem  einen block freihalten
3195 :  if z(b)=1 and b<19 then l=l-1
3200 :  b=b+1
3085 :  w=int(w)
3220 : next j
3230 next i
3310 if l<1 then 8000
3390 print: print: print "Es werden nun";l;" Bloecke freigestellt !": print
3990 :
4000 rem  ******************************
4010 rem    fuellen der dir.- bloecke
4020 rem  ******************************
4021 :
4025 b=0: rem  blockzaehler, siehe oben
4027 for e=0 to e(b)-1: rem eintr/block
4030 at=t(b): rem  alter dir-track
4035 as=s(b): ns=as: rem  alter sector
```

```
4040 if at=0 and as=0 then 7050
4050 i=0
4060 z=e*32+3: rem  stelle des zeigers
4100 r=0: rem  marke dir.-block aendern
4150 print# 1, "u1:"2;0;at;as: rem  alt
4160 print# 1, "b-p:"2,z: rem  zeiger
4170 get# 2, t$,s$: rem  zeiger forts.
4180 t=asc(t$+chr$(0))
4190 s=asc(s$+chr$(0))
4191 if t=18 then 4670: rem  bearbeitet
4192 if t=0 or t>35 or s>20 then z=1: goto 4520: rem  fileende
4200 for i=0 to 18 step 10: rem  suche nach freiem sector im sectorabstand
4210 : if z(i)=0 then 4500: rem  leer
4212 : if i=9 then goto 4230
4215 : if i>8 then i=i-9: goto 4210
4220 next i
4230 z=-1: rem  marke: spur 18 voll
4300 goto 4520: rem  rueck ohne aender.
4500 print# 1, "b-p:"2,z: rem  zeiger
4510 print# 2, chr$(18);chr$(i);: rem   zeiger verbiegen
4515 if at=18 and as=ns then r=1: rem   in den block, aus dem gelesen wurde
4520 print "       Block";right$("  "+str$(at),3);",";right$("  "+str$(as),3);
4525 if r=1 then print"  Zeiger";e+1;"geaendert": goto 4570
4530 print "  --> 18,";right$("  "+str$(ns),3)
4550 print# 1, "b-a:"0,18,ns:rem in bam
4560 print# 1, "b-f:"0,at,as: rem  alten block freigeben
4570 z(i)=1: rem block in liste belegen
4580 print# 1, "u2:"2;0;18;ns:rem ruec
4650 at=t: as=s: ns=i: rem  norm. block
4655 if z=-1 then goto 9000: rem  ok
4660 if z<>1 then z=0: goto 4100
4670 next e: rem  neuer dir.-eintrag
4680 b=b+1: goto 4027: rem  weiter mit naechstem directoryblock
6990 :
7000 rem  ******************************
7010 rem          fehlermeldungen
7020 rem  ******************************
7030 :
7050 print: print "Es waren nicht genug belegte Blocks da."
7060 print " Spur 18 ist immernoch teilweise frei!"
7090 goto 9000
7800 print:print:print"Die Diskette ist leer!"
7810 goto 9000
8000 print "Tut mir leid, da ist nichts zu machen."
8010 print "Die gesamte Spur 18 ist voll belegt !"
9000 print
9010 close 1: close 2: end
```

11.11 Vertrauen ist gut, Kontrolle ist besser

Mit dem »DISKTESTER« lassen sich in knapp zehn Minuten komplette Disketten auf schadhafte Stellen untersuchen, wobei die unbrauchbaren Sektoren als belegt markiert werden, damit die Floppy sie nicht mehr zur Datenspeicherung einsetzt. So ist es möglich, auch teilweise beschädigte Disketten noch zur Datenspeicherung heranzuziehen, ohne daß es zu Fehlermeldungen wie READ ERROR oder WRITE ERROR kommt.

Listing 11.5: Disktester

```
100 REM*DISKTESTER
105 REM*ULI LANG*
110 REM*TALSTR.10*
115 REM*8609 BISCHBERG*
120 REM*TEL.0951/67389*
125 :
130 L$="(CLR)":D$="(DOWN)":U$="(UP)":Z$="(LEFT)":REM*LOESCHEN,CRSR DOWN,CRSR UP,
CRSR BACK*
135 G$="(GRN)":O$="(ORNG)":B$="(LBLU)":T$="(CYN)":R$="(LRED)":REM*GRUEN,ORANGE,H
ELLBLAU,TUERKIS,ROT*
140 FARBSPEICHER=55296:BILDSCHIRMSPEICHER=1024
145 DIMT%(100),S%(100):REM*FEHLERHAFTE BLOCKS*
150 PRINTL$G$D$SPC(15)"DISKTESTER"D$:POKE53280,0:POKE53281,0:REM*SCWARZ*
155 PRINTO$D$"<"B$"A"O$"> ALLE    BLOCKS TESTEN"
160 PRINTD$"<"B$"B"O$"> BELEGTE BLOCKS TESTEN"
165 PRINTD$"<"B$"K"O$"> KAPUTTE BLOCKS BELEGEN"
170 GETA$
175 IFA$="A"THENTS=1:GOTO210
180 IFA$="B"THENTS=0:GOTO210
185 IFA$="K"THEN575
190 GOTO170
195 REM*****************
200 REM*DISKETTE TESTEN*
205 REM*****************
210 DIMB%(35,23),F$(8):K$=",":N$=CHR$(0):FR=0:FE=0:FT=0
215 FORI=0TO8:READF$(I):NEXT:REM*FEHLERMELDUNGEN*
220 GOSUB610:REM*AUF SPACE WARTEN*
225 OPEN15,8,15,"I"
230 PRINTL$"VALIDATE? "B$"J/N"
235 GETA$
240 IFA$="N"THEN255
245 IFA$="J"THENPRINTL$T$D$"VALIDATING...":PRINT#15,"V":GOTO255
250 GOTO235
255 OPEN3,8,3,"#":IFTSTHENOPEN5,8,5,"#"
260 PRINTL$T$"BAM WIRD GELESEN          "U$
265 PRINT#15,"U1:"3;0;18;0:GET#3,A$,A$,A$
270 IFA$<>CHR$(65)THENPRINTD$"KEIN DOS 2.6 ODER 2.5!":GOTO550
275 GET#3,A$
280 REM***********************
285 REM*ANALYSIEREN DER BAM  *
287 REM*AUFGELOEST IN B%(T,S)*
290 REM***********************
295 FORT=1TO35
300   REM*FREIE BLOCKS,SEKTOR 7-0,SEKTOR 15-8,SEKTOR 23-16*
305   GET#3,FR$:FR=FR+ASC(FR$+N$):S=0
310   IFFRTHEN320:REM*NICHT ALLE BLOCKS BELEGT*
315   FORS=0TO23:B%(T,S)=0:NEXTS:GOTO345
320   A=1:GET#3,BA$:BA=ASC(BA$+N$)
325   FORS=STOS+7
330     B%(T,S)=(BAANDA)/A:A=A+A:FT=FT+B%(T,S)
```

```
335    NEXTS
340    IFS<23THEN320
345 NEXTT
350 IFFR<>FTTHENPRINTL$R$TAB(14)"!BAM-FEHLER!":GOTO550
355 GOSUB635
360 REM*************************
365 REM*START DER TESTROUTINE*
370 REM*************************
375 IFTS=0THENFR=683-FR:REM*VOLLE BLOCKS*
380 IFTS=0THEN400
385 REM*170=%10101010 , 85=%01010101*
390 PRINT#15,"B-P:"5;1:FORI=1TO128:PRINT#5,CHR$(170)CHR$(85);:NEXT
395 REM*PUFFER BRAUCHT NUR EINMAL GEFUELLT ZU WERDEN*
400 FORT=1TO35:REM*SPURZAEHLER*
405    IFT<18THENMS=20:GOTO425
410    IFT<25THENMS=18:GOTO425
415    IFT<31THENMS=17:GOTO425
420    MS=16
425    FORS=0TOMS:REM*BLOCKZAEHLER*
430       ONTS*2+B%(T,S)GOTO485,435,440
435       PRINT#15,"U1:"3;0;T;S:GOTO445:REM*BLOCK LESEN*
440       PRINT#15,"B-W:"5;0;T;S:REM*BUFFER AUF DISK SCHREIBEN*
445       AO=40*(20-S)+T+3:REM*ADRESSOFFSET*
450       INPUT#15,F:IFF<20THENP=32:GOTO480
455       IFF<29THENF$=F$(F-20):GOTO465
460       F$=""
465       PRINTR$F;F$" T:"T;Z$" S:"S;Z$" "U$
470       POKEFA+AO,10:P=81+6*B%(T,S):REM*ROTES"●,○"
475       T%(FE)=T:S%(FE)=S:FE=FE+B%(T,S)
480       POKEBI+AO,P
485    NEXTS
490 NEXTT:CLOSE5:CLOSE3
495 IFFE=0THENPRINT"      !KEINE ZU BELEGENDEN FEHLER!";:GOTO550
500 REM*******************************
505 REM*BELEGEN DER KAPUTTEN BLOCKS*
510 REM*******************************
515 PRINTL$T$" TEST ZUENDE, BELEGEN UND KENNZEICHNEN"
520 PRINTTAB(8)"DER FEHLERHAFTEN BLOCKS"
525 FORI=0TOFE-1
530    PRINT#15,"B-A:"0;T%(I);S%(I):REM*IN BAM BELEGEN*
535 NEXT
540 OPEN3,8,3,"@:.DISKFEHLER,U,W":PRINT#3,CHR$(FE-1);
545 FORI=0TOFE-1:PRINT#15,T%(I)CHR$(13)S%(I):NEXT
550 CLOSE3:CLOSE15:END
555 :
560 REM********************************
565 REM*WIEDERBELEGUNG NACH VALIDATE*
570 REM********************************
575 GOSUB610
580 PRINTL$T$"WIEDERBELEGEN DER SCHADHAFTEN BLOCKS"
585 OPEN15,8,15,"I":OPEN5,8,5,".DISKFEHLER,U,R"
590 GET#5,A$:FE=ASC(A$)
595 FORI=0TOFE:INPUT#5,T%(I),S%(I):PRINTR$"T:"T%(I),"S:"S%(I):NEXT:CLOSE5
600 FORI=0TOFE:PRINT#15,"B-A:"0;T%(I);S%(I):NEXT:CLOSE15:END
605 :
610 PRINTL$TAB(9)O$"TESTDISKETTE  EINLEGEN":PRINTTAB(17)"<"B$"SPACE"O$">"
615 GETA$:IFA$<>" "THEN615
620 RETURN
625 :
630 REM*BAM DARSTELLEN*
635 PRINTL$;
640 FORS=20TO0STEP-1
645 PRINTO$S;TAB(4);
```

```
650    FORT=1TO35
655      IFT<18THENMS=20:GOTO675
660      IFT<25THENMS=18:GOTO675
665      IFT<31THENMS=17:GOTO675
670      MS=16
675      IFS>MSTHEN690
680      IFB%(T,S)=0THENPRINTB$"-";:GOTO690
685      PRINTG$"*";
690    NEXTT:PRINT
695 NEXTS:PRINTTAB(4);
700 FORT=1TO35:PRINTRIGHT$(STR$(T),1);:NEXT
705 PRINT:RETURN
710 REM*FEHLERMELDUNGEN*
715 DATA"KEIN HEADER       ","KEIN SYNC CHAR    ","KEIN BLOCK VORH.  "
720 DATA"PRUEFSUMME DATEN  ","BITMUSTERFEHLER   ","SCHREIBFEHLER     "
725 DATA"SCHREIBSCHUTZ     ","PRUEFSUMME HEADER","BLOCK ZU LANG     "
```

Nach dem Starten des Programms muß man entscheiden, ob man nur die vollen Blocks (durch Lesen) oder alle Blocks (durch Lesen und Zurückschreiben) testen will, oder ob man die schon bei vorherigen Tests als fehlerhaft erkannten Blocks nach einem VALIDATE wieder belegen will. Danach wird gefragt, ob der Befehl VALIDATE an die Floppy gesandt werden soll. Dies ist nur bei Disketten sinnvoll, auf denen die Daten öfters geändert werden und die keine relativen Dateien oder ähnliche Formate wie GEOS enthalten.

Bei Disketten mit den oben genannten Dateien, von denen man nicht sicher weiß, ob die beschriebenen Blöcke auch in der BAM als belegt gekennzeichnet sind, sollte die Option »Leere Blocks testen« auf keinen Fall angewandt werden, da sonst Daten verlorengehen könnten.

Es folgt nach kurzer Lese- und Decodierzeit die symbolische Belegung der Diskette auf dem Bildschirm, auf der die belegten Sektoren als »–« und die nicht belegten als »+« gekennzeichnet sind. Wie kommt es dazu? Nun, jede Spur belegt in der BAM (Block 18/0, Bytes 4–143) 4 Byte. Zuerst kommt die Anzahl der freien Blöcke auf der Spur, im Programm bezeichnet mit FR$. Dann kommt nacheinander die Belegung der Sektoren 7–0, 15–8 und 23–16. Ein 1-Bit repräsentiert einen freien Block, ein 0-Bit einen besetzten.

Dies wird nun in der Routine »Analysieren der BAM« aufgelöst und in B%(T,S) festgehalten, wobei T die Spur (Track) und S den Sektor enthält. Die darauffolgende Testroutine beschränkt sich darauf, die Spuren und Sektoren einzeln durchzuzählen und in Abhängigkeit von B$(T,S) und der Variablen TS (0 = alle Blocks, 1 = nur volle Blocks) entweder den Block zu lesen oder ihn mit dem einmal aufgefüllten Puffer zu beschreiben.

Danach wird der Fehlerkanal auf die Fehlernummer abgefragt, da dies schneller geht als immer die ganze Fehlermeldung zu holen. Tritt ein Fehler auf, so wird dieser angezeigt und auf dem Bildschirm als roter Punkt in der BAM markiert; gleichzeitig erhöht sich die Variable FE. Tritt kein Fehler auf, so wird die Kennzeichnung für den gerade getesten Block am Bildschirm gelöscht.

Sind alle Blocks getestet, werden die eventuell vorhandenen defekten Blocks auf der Diskette belegt und in dem Format »Spur,CHR$(13),Block,CHR$(13)« in dem USR-File ».DISKFEHLER« gespeichert. Das erste Byte entspricht in diesem Fall der Anzahl der gefundenen Fehler. Die Routine »Kaputte Blocks belegen« hat nun nichts weiter zu tun, als dieses File zu lesen und die Blocks zu belegen.

Zehn Minuten für das Überprüfen einer Diskette mögen zwar immer noch recht lang anmuten, aber wenn man bedenkt, wieviel Ärger ein fehlerhafter Sektor verursachen kann, dann rentiert es sich wirklich, diese Zeit zu investieren.

11.12 Dateien kopieren

Was soll ein gutes Kopierprogramm leisten? Nun, zunächst einmal soll es kopieren. Dazu muß man auswählen können, was kopiert werden soll. Fehler beim Kopieren sollten möglichst ohne Programmabsturz und völligen Neubeginn zu beheben sein. Gerade hier liegt die Stärke der beiden File-Kopierprogramme, die Sie im folgenden erhalten.

11.12.1 Kopieren mit Komfort: Super-Copy

Bis zu 32 Programme mit insgesamt 234 Blöcken lassen sich mit »Super-Copy« in einem Rutsch problemlos kopieren. Das Besondere: »Super-Copy« ist so verträglich, daß es mit und ohne Speeder keinerlei Probleme aufwirft. Bei sehr leistungsfähigen Speedern, die um ein Vielfaches schneller sind als selbst die besten Kopierprogramme mit Software-Beschleuniger, ist somit »Super-Copy« bei weitem das effektivste Programm. Und der Bedienungskomfort sucht, obwohl dieses Programm schon einige Jahre alt ist, immer noch seinesgleichen!
Nach dem Programmstart mit RUN meldet sich das Programm mit einem Menü:

1. Directory
2. Kopieren
3. Formatieren
4. Scratchen
5. Validieren
6. Ende

Durch Druck auf eine Ziffer wählen Sie die entsprechende Funktion. Übrigens können Sie im Programm immer dann, wenn Sie irgendeine Taste drücken müssen, mit <←> in dieses Menü zurückkehren. Gehen wir nun die einzelnen Funktionen einmal durch.
Zur Funktion »Directory« ist nicht viel zu sagen. Es erscheint das Verzeichnis aller Files auf dieser Diskette. Beim »Formatieren« müssen Sie den Diskettennamen und die ID – wie üblich durch ein Komma getrennt – angeben. Eine ID ist nur bei einer neuen Diskette wichtig. Verzichten Sie darauf, werden zwar alle Einträge im Directory gelöscht, aber es entfällt das Neuformatieren der einzelnen Spuren. »Validieren« (Gültigkeitskontrolle) entspricht dem Basic-Befehl OPEN1,8,15,"V":CLOSE1. Entscheiden Sie sich für »Kopieren«, werden Sie aufgefordert, die Quelldiskette einzulegen. Nach Tastendruck erscheinen nun die Namen der Programme. Files, die kopiert werden sollen, kennzeichnen Sie mit der J-Taste, die anderen mit <N>. Relative Files können nicht kopiert

werden, daher erscheint eine Fehlermeldung, wenn Sie versuchen, solche Files mit <J> zu markieren.
Das Programm kann maximal 32 Namen speichern. Wenn Sie mehr als 32 Files kopieren wollen, erscheint die Fehlermeldung »Kopierliste voll«. Sie können nun die bisher markierten Programme kopieren und nach Abschluß einen neuen Programmdurchlauf starten. Haben Sie Ihre Auswahl beendet, gibt das Programm an, wie viele Blöcke insgesamt zu kopieren sind, damit Sie genügend Platz auf der Zieldiskette bereitstellen können. Ein neues Menü erscheint:

```
1. Directory
2. Formatieren
3. Validieren

*** SPACE ***
für weiter
```

Sie können nun in aller Ruhe eine Zieldiskette aussuchen, eventuell noch formatieren etc. Sie kommen in jedem Fall in dieses Menü zurück. Sind alle Vorbereitungen abgeschlossen, drücken Sie <Leertaste>, um mit dem Kopieren fortzufahren.
Das Programm fordert nun auf, die Quelldiskette einzulegen und liest die vorher markierten Programme ein. Sollte dabei ein Fehler auftreten, weil Sie zum Beispiel die falsche Diskette eingelegt haben, wird eine entsprechende Meldung ausgegeben und gefragt, ob dieses File übersprungen oder ein neuer Versuch unternommen werden soll. Auch Lesefehler des Laufwerks werden in dieser Weise gehandhabt. In einem Durchgang können maximal 234 Blöcke eingelesen werden; ist noch mehr zu kopieren, wird das Einlesen abgebrochen.
Jetzt müssen Sie angeben, ob Sie fortlaufend oder einzeln kopieren möchten. Fortlaufend bedeutet, daß die Files der Reihe nach auf dieselbe Diskette geschrieben werden. Beim Einzelkopieren dagegen springt das Programm nach jedem Schreibvorgang wieder in ein Menü, und Sie haben die Möglichkeit, die Diskette zu wechseln, ein Directory anzusehen, zu formatieren oder zu validieren. Außerdem können Sie das zuletzt kopierte Programm noch einmal auf eine andere Diskette kopieren. Das jeweils nächste File wird vor dem Schreiben angezeigt, damit Sie die richtige Zieldiskette einlegen können.
Haben Sie Ihre Wahl getroffen, läuft der Schreibvorgang in der beschriebenen Art und Weise ab. Schreibfehler werden wie Lesefehler behandelt, d.h. es wird abgefragt, ob ein neuer Versuch gestartet oder das File übersprungen werden soll.
Sind alle Programme kopiert, erscheint die Meldung »KOPIE FERTIG«. Sind aber nach dem ersten Schreib-Lese-Durchgang noch weitere Programme zu kopieren, fährt das Programm mit der Aufforderung zum Einlegen der Quelldiskette fort.
»Scratchen«, also das Löschen von Programmen, gehört zu den angenehmsten Funktionen, die »Super-Copy« zu bieten hat. Seien Sie aber vorsichtig, sonst haben Sie bald überhaupt keine Programme mehr.
Der Ablauf ist ähnlich wie beim Kopieren. Doch anstatt die Files zum Kopieren zu kennzeichnen, werden sie nun zum Löschen markiert. Sollten Sie einen Fehler gemacht haben, ist allerdings noch nichts verloren, denn am Ende des Markierungsvorgangs müssen Sie noch einmal ausdrücklich durch Drücken der Leertaste bestätigen, daß es Ihnen ernst ist.
Zum Programmaufbau: Das Programm ist im wesentlichen modular aufgebaut, d.h. es besteht aus einzelnen Blöcken, die von den verschiedenen Menüs aus angesprungen werden. Daher ist

es nicht nötig, häufig vorkommende Programmteile sowie Tasten- oder Fehlerabfragen ständig neu zu schreiben. Auch der Teil zum Markieren der Programme ist beim Scratchen und beim Kopieren identisch. Ein Flag, also eine Speicherstelle, deren Inhalt etwas »signalisieren« soll, entscheidet darüber, welche Texte ausgegeben und wohin zurückgesprungen werden soll. Eine solche Programmiertechnik trägt dazu bei, Programme ebenso kurz wie übersichtlich zu halten.

Als Listing 11.6 finden Sie nun einen Leckerbissen, den wir alle dem Programmautor, Herrn Dietrich Weineck, zu verdanken haben: den Quelltext zu »Super-Copy«! Dieses Programm ist so geschickt realisiert worden, daß sich jeder an Maschinensprache Interessierte eine Scheibe davon abschneiden kann! Der vorliegende Quelltext bezieht sich zwar auf eine erweiterte Programmversion, die für den Betrieb mit zwei Laufwerken gedacht ist, aber Sie werden darin keine Information über die »tatsächliche« Fassung vermissen.

Listing 11.6: Quelltext zu SUPER-COPY

```
0                  .BA $0810
1                  .MC $7010
2   FILENR         .DE $02C0
3   FNAMEL         .DE $1500
4   NAMELIST       .DE FNAMEL+$100
5   COPYSTART      .DE FNAMEL+$300
6   FILETYP        .DE FNAMEL+$20
7   FILEADR        .DE FNAMEL+$40
8   ENDADRL        .DE FNAMEL+$60
9   ENDADRH        .DE FNAMEL+$80
10  FILEL          .DE FNAMEL+$A0
11  FILEANZ        .DE FILENR+1
12  FILECOP        .DE FILENR+2
13  CONTFLG        .DE FILENR+3
14  SRFLAG         .DE FILENR+4
15  BLKSUML        .DE FILENR+5
16  BLKSUMH        .DE FILENR+6
17  QUELL          .DE FILENR+7
18  ZIEL           .DE FILENR+8
19  DEV            .DE FILENR+9
20  BUFFER         .DE $0340
21  ACPTR          .DE $FFA5
22  CHRIN          .DE $FFCF
23  CHROUT         .DE $FFD2
24  CHKIN          .DE $FFC6
25  CHKOUT         .DE $FFC9
26  CIOUT          .DE $FFA8
27  CLALL          .DE $FFE7
28  CLOSE          .DE $FFC3
29  CLRCHN         .DE $FFCC
30  GETIN          .DE $FFE4
31  LISTEN         .DE $FFB1
32  OPEN           .DE $FFC0
33  SETLFS         .DE $FFBA
34  SETNAM         .DE $FFBD
35  TALK           .DE $FFB4
36  TKSA           .DE $FF96
37  UNLSN          .DE $FFAE
38  UNTLK          .DE $FFAB
39                 .OS
40                 ;
41                 ;
42                 LDA #$0B
43                 STA $D020
44                 STA $D021
45                 LDA #$01
46                 STA $0286
47                 LDA #$80
48                 STA $028A
49                 LDA #$08
50                 STA QUELL
51                 STA ZIEL
52  START          LDA #$0B
53                 STA $D020
54                 LDA #L,TEXT1
55                 LDY #H,TEXT1
56                 JSR TEXTAUS
57  START0         JSR TAST
58                 CMP #$31
59                 BCC START0
60                 CMP #$38
61                 BCS START0
62                 JSR CHROUT
63                 CMP #$31
64                 BNE START1
65                 JSR DIRECTORY
66                 JMP START
67  START1         CMP #$32
68                 BNE START2
69                 JMP COPY
70  START2         CMP #$33
71                 BNE START3
72                 JSR FORMAT
73                 JMP START
74  START3         CMP #$34
75                 BNE START4
76                 JMP SCRATCH
77  START4         CMP #$35
78                 BNE START5
79                 JSR VALIDATE
80                 JMP START
81  START5         CMP #$36
82                 BNE START6
83                 JSR DEVICE
84                 JMP START
85  START6         JMP $A474
86                 ;
87                 ;
```

```
88   DIRECTORY   JSR DEVICE?
89               LDA #$93
90               JSR CHROUT
91               LDA DEV
92               TAX
93               LDY #$00
94               JSR SETLFS
95               LDA #$01
96               LDX #L,NAME1
97               LDY #H,NAME1
98               JSR SETNAM
99               JSR OPEN
100              BCC DIREC0
101              JMP DIREC4
102  DIREC0      LDA DEV
103              JSR TALK
104              LDA #$00
105              JSR TKSA
106              JSR ACPTR
107              JSR ACPTR
108              JMP DIREC3
109  DIREC1      JSR ACPTR
110              STA *$FB
111              JSR ACPTR
112              LDX *$FB
113              JSR $BDCD
114              JSR SPACE
115  DIREC2      JSR ACPTR
116              BEQ DIREC3
117              JSR CHROUT
118              CLC
119              BCC DIREC2
120  DIREC3      JSR RETURN
121              JSR ACPTR
122              JSR ACPTR
123              LDA *$90
124              BEQ DIREC1
125              JSR UNTLK
126  DIREC4      LDA DEV
127              JSR CLOSE
128              JSR FEHLER
129              JSR TASTE
130              RTS
131              ;
132              ;
133  COPY        JSR MARKIEREN
134              LDA #L,TEXT24
135              LDY #H,TEXT24
136              JSR TEXTAUS
137              LDA #$FF
138              STA CONTFLG
139              JSR MENU2
140              LDA QUELL
141              STA DEV
142              JSR READ
143              LDA QUELL
144              CMP ZIEL
145              BNE COPY1
146              JSR MENU1
147  COPY1       LDA ZIEL
148              STA DEV
149              JSR RETURN
150              JSR WRITE
151              LDA QUELL
152              STA DEV
153              JSR READ1
154              JMP COPY1
155              ;
156              ;
157  FORMAT      JSR DEVICE?
158              LDA #L,TEXT16
159              LDY #H,TEXT16
160              JSR TEXTAUS
161              LDX #$00
162  FORM1       JSR CHRIN
163              STA BUFFER,X
164              INX
165              CMP #$0D
166              BNE FORM1
167              LDA #$00
168              DEX
169              STA BUFFER,X
170  FORM2       LDA #'N'
171              STA BUFFER-3
172              LDA #'0'
173              STA BUFFER-2
174              LDA #':'
175              STA BUFFER-1
176              LDA #$0F
177              LDX DEV
178              TAY
179              JSR SETLFS
180              LDX #$03
181  FORM3       LDA BUFFER-3,X
182              BEQ FORM4
183              INX
184              BNE FORM3
185  FORM4       TXA
186              LDX #L,BUFFER-3
187              LDY #H,BUFFER-3
188              JSR SETNAM
189              JSR OPEN
190              JSR FEHLER1
191              PHP
192              LDA #$0F
193              JSR CLOSE
194              PLP
195              BCS FORM5
196              RTS
197  FORM5       LDA #L,TEXT17
198              LDY #H,TEXT17
199              JSR TEXTAUS
200  FORM6       JSR TASTE
201              CMP #'J'
202              BEQ FORM2
203              CMP #'N'
204              BNE FORM6
205              JMP START
206              ;
207              ;
208  SCRATCH     JSR DEVICE?
209              LDA #$93
210              JSR CHROUT
211              LDA #$07
212              STA $D020
213              LDA #L,TEXT21
214              LDY #H,TEXT21
215              JSR TEXTAUS
216              LDA #$80
217              STA SRFLAG
218              JSR MARK1
219              LDA DEV
```

```
220                    JSR CLOSE
221                    LDA FILEANZ
222                    BNE SCR1
223                    JMP START
224     SCR1           JSR BLKSUMME
225                    LDA #L,TEXT25
226                    LDY #H,TEXT25
227                    JSR TEXTAUS
228                    JSR READ
229     SCR2           LDA #'S'
230                    STA BUFFER-3
231                    LDA #'0'
232                    STA BUFFER-2
233                    LDA #':'
234                    STA BUFFER-1
235                    TXA
236                    CLC
237                    ADC #$03
238                    LDX #L,BUFFER-3
239                    LDY #H,BUFFER-3
240                    JSR SETNAM
241                    LDA #$0F
242                    LDX DEV
243                    TAY
244                    JSR SETLFS
245                    JSR OPEN
246                    LDA #$0F
247                    JSR CLOSE
248                    LDX FILENR
249                    INX
250                    CPX FILEANZ
251                    BCS SCREND
252                    INC FILENR
253                    JSR SCRM
254                    JMP SCR2
255     SCREND         JMP START
256                    ;
257                    ;
258     VALIDATE       JSR DEVICE?
259                    LDA #'V'
260                    STA NAME2
261                    LDA #L,TEXT10
262                    LDY #H,TEXT10
263                    JSR TEXTAUS
264                    LDA DEV
265                    JSR DISKCMD
266                    LDA #'I'
267                    STA NAME2
268                    JSR FEHLER
269                    RTS
270                    ;
271                    ;
272     MARKIEREN      LDA #$93
273                    JSR CHROUT
274                    LDA #$00
275                    STA SRFLAG
276                    LDA QUELL
277                    STA DEV
278                    LDA #L,TEXT11
279                    LDY #H,TEXT11
280                    JSR TEXTAUS
281     MARK1          JSR TASTE
282                    JSR RETURN
283                    JSR RETURN
284                    JSR DISKCMD
285                    JSR FEHLER
286                    BCC MARK1.1
287                    JSR TASTE
288                    PLA
289                    PLA
290                    JMP START
291     MARK1.1        LDA DEV
292                    TAX
293                    LDY #$00
294                    JSR SETLFS
295                    LDA #$01
296                    LDX #L,NAME1
297                    LDY #H,NAME1
298                    JSR SETNAM
299                    JSR OPEN
300                    LDA DEV
301                    JSR TALK
302                    LDA #$00
303                    JSR TKSA
304                    LDY #$04
305     MARK2          JSR ACPTR
306                    DEY
307                    BNE MARK2
308                    JSR ACPTR
309                    STA *$FB
310                    JSR ACPTR
311                    LDX *$FB
312                    JSR $BDCD
313                    JSR SPACE
314     MARK3          JSR ACPTR
315                    BEQ MARK4
316                    JSR CHROUT
317                    CLC
318                    BCC MARK3
319     MARK4          JSR RETURN
320                    JSR RETURN
321                    JSR ACPTR
322                    JSR ACPTR
323                    LDY #$00
324                    STY FILEANZ
325     MARK5          JSR ACPTR
326                    STA FILENR
327                    JSR ACPTR
328                    STA FILECOP
329                    LDX FILENR
330                    JSR $BDCD
331                    JSR SPACE
332                    LDY #$00
333     MARK6          JSR ACPTR
334                    JSR CHROUT
335                    STA BUFFER,Y
336                    BEQ MARK7
337                    INY
338                    BNE MARK6
339     MARK7          JSR ACPTR
340                    JSR ACPTR
341                    LDA *$90
342                    BEQ MARK9
343                    BIT SRFLAG
344                    BPL MARK8
345                    RTS
346     MARK8          JMP MARK23
347     MARK9          LDA FILECOP
348                    BNE MARK10
349                    LDA FILENR
350                    CMP #$EB
351                    BCC MARK12
```

```
352   MARK10     LDA #L,TEXT2        418                 CLC
353              LDY #H,TEXT2        419                 BCC MARK22
354              JSR TEXTAUS         420   MARK21        LDA #L,TEXT7
355   MARK11     JMP MARK22          421                 LDY #H,TEXT7
356   MARK12     LDA FILEANZ         422                 JSR TEXTAUS
357              CMP #$20            423   MARK22        JSR RETURN
358              BCC MARK13          424                 JMP MARK5
359              LDA #L,TEXT3        425   MARK23        JSR UNTLK
360              LDY #H,TEXT3        426                 LDA DEV
361              JSR TEXTAUS         427                 JSR CLOSE
362              CLC                 428                 LDA FILEANZ
363              BCC MARK8           429                 BNE BLKSUMME
364   MARK13     LDA #$00            430                 PLA
365              STA *$08            431                 PLA
366              LDA #$20            432                 JMP START
367              STA *$D3            433                 ;
368              LDA #L,TEXT4        434                 ;
369              LDY #H,TEXT4        435   BLKSUMME      LDX #$00
370              JSR TEXTAUS         436                 STX BLKSUMH
371   MARK14     JSR TASTE           437                 STX BLKSUML
372              CMP #$4E            438                 JSR RETURN
373              BEQ MARK21          439                 JSR RETURN
374              CMP #'J'            440                 LDA #$12
375              BNE MARK14          441                 JSR CHROUT
376              LDA #L,TEXT5        442                 CLC
377              LDY #H,TEXT5        443   BLK1          LDA FILEL,X
378              JSR TEXTAUS         444                 ADC BLKSUML
379              LDA FILEANZ         445                 STA BLKSUML
380              JSR MEMSTART        446                 LDA #$00
381              LDX #$00            447                 ADC BLKSUMH
382   MARK15     INX                 448                 STA BLKSUMH
383              LDA BUFFER,X        449                 INX
384              CMP #$22            450                 CPX FILEANZ
385              BNE MARK15          451                 BCC BLK1
386              STX FILECOP         452                 LDX BLKSUML
387              INX                 453                 JMP $BDCD
388   MARK16     LDA BUFFER,X        454                 ;
389              CMP #$22            455                 ;
390              BEQ MARK17          456   MENU1         LDA #L,TEXT9
391              STA ($41),Y         457                 LDY #H,TEXT9
392              INX                 458                 JSR TEXTAUS
393              INY                 459                 BIT CONTFLG
394              BNE MARK16          460                 BMI MENU1.0
395   MARK17     TXA                 461                 LDA #L,TEXT27
396              LDY FILEANZ         462                 LDY #H,TEXT27
397              CLC                 463                 JSR TEXTAUS
398              SBC FILECOP         464   MENU1.0       LDA #L,TEXT26
399              STA FNAMEL,Y        465                 LDY #H,TEXT26
400              LDA FILENR          466                 JSR TEXTAUS
401              STA FILEL,Y         467   MENU1.1       JSR TASTE
402   MARK18     LDA BUFFER,X        468                 CMP #$20
403              BNE MARK19          469                 BNE MENU1.2
404              BIT SRFLAG          470                 RTS
405              BMI MARK20          471   MENU1.2       CMP #$32
406              LDA #L,TEXT6        472                 BNE MENU1.3
407              LDY #H,TEXT6        473                 JSR VALIDATE
408              JSR TEXTAUS         474                 JMP MENU1
409              JMP MARK22          475   MENU1.3       CMP #$31
410   MARK19     CMP #$53            476                 BNE MENU1.4
411              BEQ MARK20          477                 JSR DIRECTORY
412              CMP #$50            478                 JMP MENU1
413              BEQ MARK20          479   MENU1.4       CMP #$33
414              INX                 480                 BNE MENU1.5
415              BNE MARK18          481                 JSR FORMAT
416   MARK20     STA FILETYP,Y       482                 JMP MENU1
417              INC FILEANZ         483   MENU1.5       CMP #$34
```

```
484                   BNE MENU1.1
485                   BIT CONTFLG
486                   BMI MENU1.1
487                   PLA
488                   PLA
489                   JMP WRITE1
490                   ;
491                   ;
492      MENU2        LDA #L,TEXT8
493                   LDY #H,TEXT8
494                   JSR TEXTAUS
495      MENU2.1      JSR TASTE
496                   CMP #$31
497                   BEQ MENU2.2
498                   CMP #$32
499                   BNE MENU2.1
500                   LDA #$00
501                   .BY $2C
502      MENU2.2      LDA #$FF
503                   STA CONTFLG
504                   RTS
505                   ;
506                   ;
507      READ         LDX #$00
508                   STX FILENR
509      READ1        LDA #H,COPYSTART
510                   STA FILEADR,X
511                   LDA #$00
512                   STA FILECOP
513                   BIT SRFLAG
514                   BPL READ2
515                   LDA #L,TEXT22
516                   LDY #H,TEXT22
517                   JSR TEXTAUS
518      READ1.5      JSR TASTE
519                   CMP #$20
520                   BNE READ1.5
521      SCRM         LDA #L,TEXT23
522                   LDY #H,TEXT23
523                   JSR TEXTAUS
524                   CLC
525                   BCC READ5
526      READ2        TXA
527                   BEQ READ4
528                   LDA QUELL
529                   CMP ZIEL
530                   BNE READ4
531                   LDA #L,TEXT11
532                   LDY #H,TEXT11
533                   JSR TEXTAUS
534      READ3        JSR TASTE
535      READ4        LDA #L,TEXT12
536                   LDY #H,TEXT12
537                   JSR TEXTAUS
538                   LDA #$06
539                   STA $D020
540      READ5        LDY FILENR
541                   LDX FNAMEL,Y
542                   LDA FILENR
543                   JSR MEMSTART
544      READ6        LDA ($41),Y
545                   JSR CHROUT
546                   INY
547                   DEX
548                   BNE READ6
549                   JSR PRINTBLK
550                   LDX FILENR
551                   LDA FNAMEL,X
552                   STA *$FB
553                   LDA FILENR
554                   JSR MEMSTART
555                   LDX #$00
556      READ7        LDA ($41),Y
557                   STA BUFFER,X
558                   INY
559                   INX
560                   DEC *$FB
561                   BNE READ7
562                   BIT SRFLAG
563                   BPL READ8
564                   RTS
565      READ8        LDY #$00
566      READ9        LDA LESEN,Y
567                   STA BUFFER,X
568                   INY
569                   INX
570                   CPY #$04
571                   BCC READ9
572                   LDY FILENR
573                   LDA FILETYP,Y
574                   STA BUFFER-3,X
575                   TXA
576                   LDX #$40
577                   LDY #$03
578                   JSR SETNAM
579                   LDA #$02
580                   LDX DEV
581                   TAY
582                   JSR SETLFS
583                   JSR OPEN
584                   LDX FILENR
585                   LDA FILEADR,X
586                   LDY #$00
587                   STY *$FB
588                   STA *$FC
589                   LDX #$02
590                   JSR CHKIN
591      READ10       JSR ACPTR
592                   JSR RW1
593                   LDX *$90
594                   BEQ READ10
595                   JSR FEHLER
596                   PHP
597                   JSR CLRCHN
598                   LDA #$02
599                   JSR CLOSE
600                   PLP
601                   BCC READ11
602                   JSR FEHLER10
603                   BCS READ12
604      READ11       LDX FILENR
605                   SEC
606                   LDA *$FB
607                   SBC #$01
608                   STA ENDADRH,X
609                   LDA *$FC
610                   SBC #$00
611                   STA ENDADRL,X
612                   INX
613                   CPX FILEANZ
614                   BCS READEND
615                   CLC
```

```
616                ADC FILEL,X
617                BCS READEND
618                INC FILENR
619                INC FILECOP
620                LDA *$FC
621                CLC
622                ADC #$01
623                STA FILEADR,X
624   READ12       LDA #$0F
625                JSR CLOSE
626                JMP READ4
627   READEND      RTS
628                ;
629                ;
630   WRITE        LDA FILENR
631                SEC
632                SBC FILECOP
633                STA FILENR
634                LDA QUELL
635                CMP ZIEL
636                BNE WRITE2
637   WRITE1       LDA #L,TEXT13
638                LDY #H,TEXT13
639                JSR TEXTAUS
640                JSR TASTE
641   WRITE2       JSR DISKCMD
642                LDA #$05
643                STA $D020
644                LDA #L,TEXT14
645                LDY #H,TEXT14
646                JSR TEXTAUS
647                LDY FILENR
648                LDX FNAMEL,Y
649                LDA FILENR
650                JSR MEMSTART
651                NOP
652   WRITE3       LDA ($41),Y
653                NOP
654                JSR CHROUT
655                INY
656                DEX
657                BNE WRITE3
658                JSR PRINTBLK
659                LDX FILENR
660                LDA FNAMEL,X
661                STA *$FB
662                LDA FILENR
663                JSR MEMSTART
664                LDX #$00
665   WRITE4       LDA ($41),Y
666                STA BUFFER,X
667                INX
668                INY
669                DEC *$FB
670                BNE WRITE4
671                LDY #$00
672   WRITE5       LDA SCHREIB,Y
673                STA BUFFER,X
674                INY
675                INX
676                CPY #$04
677                BCC WRITE5
678                LDY FILENR
679                LDA FILETYP,Y
680                STA BUFFER-3,X
681                TXA
682                LDX #$40
683                LDY #$03
684                JSR SETNAM
685                LDA #$02
686                LDX DEV
687                TAY
688                JSR SETLFS
689                JSR OPEN
690                LDX FILENR
691                LDY #$00
692                LDA FILEADR,X
693                STY *$FB
694                STA *$FC
695                LDA ENDADRH,X
696                STA *$FD
697                LDA ENDADRL,X
698                STA *$FE
699                LDX #$02
700                JSR CHKOUT
701   WRITE6       JSR RW4
702                JSR CIOUT
703                LDA *$FD
704                CMP *$FB
705                LDA *$FE
706                SBC *$FC
707                BCS WRITE6
708                JSR CLRCHN
709                LDA #$02
710                JSR CLOSE
711                JSR FEHLER
712                BCC WRITE7
713                JSR FEHLER10
714                BCS WRITE9
715   WRITE7       BIT CONTFLG
716                BMI WRITE8
717                JMP WRITE12
718   WRITE8       DEC FILECOP
719                BMI WRITE10
720                INC FILENR
721   WRITE9       JMP WRITE2
722   WRITE10      INC FILENR
723                LDX FILENR
724                CPX FILEANZ
725                BCS WRITE11
726                JSR RETURN
727                RTS
728   WRITE11      LDA #L,TEXT15
729                LDY #H,TEXT15
730                JSR TEXTAUS
731                JSR TASTE
732                PLA
733                PLA
734                JMP START
735   WRITE12      JSR MENU1
736                DEC FILECOP
737                BPL WRITE13
738                JMP WRITE10
739   WRITE13      INC FILENR
740                JSR PRNAME
741                JMP WRITE1
742                ;
743                ;
744   MEMSTART     LDY #$00
745                ASL A
746                ASL A
747                STY *$42
```

```
748                     ASL A                  814                 STA *$90
749                     ROL *$42               815                 JSR OPEN
750                     ASL A                  816                 LDA #$0F
751                     ROL *$42               817                 JMP CLOSE
752                     STA *$41               818                 ;
753                     LDA *$42               819                 ;
754                     CLC                    820   GETBYTED      JSR ACPTR
755                     ADC #H,NAMELIST        821                 AND #$0F
756                     STA *$42               822                 ASL A
757                     LDY #$00               823                 ASL A
758                     RTS                    824                 ASL A
759                     ;                      825                 ASL A
760                     ;                      826                 STA *$57
761   RWDISK            SEI                    827                 JSR ACPTR
762                     LDY #$34               828                 AND #$0F
763                     STY *$01               829                 ORA *$57
764                     LDY #$00               830                 RTS
765                     RTS                    831                 ;
766   RW1               JSR RWDISK             832                 ;
767                     STA ($FB),Y            833   PRBYT         PHA
768   RW2               INC *$FB               834                 LSR A
769                     BNE RW3                835                 LSR A
770                     INC *$FC               836                 LSR A
771   RW3               LDY #$37               837                 LSR A
772                     STY *$01               838                 JSR PRBYT1
773                     CLI                    839                 PLA
774                     RTS                    840   PRBYT1        AND #$0F
775   RW4               JSR RWDISK             841                 CLC
776                     LDA ($FB),Y            842                 ADC #$30
777                     JMP RW2                843                 JMP CHROUT
778                     ;                      844                 ;
779                     ;                      845                 ;
780   PRNAME            LDA #L,TEXT19          846   FEHLER        LDA #$0F
781                     LDY #H,TEXT19          847                 LDX DEV
782                     JSR TEXTAUS            848                 TAY
783                     LDA FILENR             849                 JSR SETLFS
784                     ASL A                  850                 LDA #$00
785                     ASL A                  851                 JSR SETNAM
786                     ASL A                  852                 JSR OPEN
787                     ASL A                  853                 JSR FEHLER1
788                     PHP                    854                 PHP
789                     LDX FILENR             855                 LDA #00
790                     LDY FNAMEL,X           856                 STA *$90
791                     TAX                    857                 LDA #$0F
792   PRN1              LDA NAMELIST,X         858                 JSR CLOSE
793                     PLP                    859                 PLP
794                     PHP                    860                 RTS
795                     BCC PRN2               861   FEHLER1       LDX #$0F
796                     LDA NAMELIST+$100,X    862                 JSR CHKIN
797   PRN2              JSR CHROUT             863                 JSR GETBYTED
798                     INX                    864                 CMP #$01
799                     DEY                    865                 PHP
800                     BNE PRN1               866                 BCC FEHLER2
801                     PLP                    867                 PHA
802                     RTS                    868                 JSR RETURN
803                     ;                      869                 JSR RETURN
804                     ;                      870                 LDA #$02
805   DISKCMD           LDA #$0F               871                 STA $D020
806                     LDX DEV                872                 PLA
807                     TAY                    873                 JSR PRBYT
808                     JSR SETLFS             874   FEHLER2       JSR ACPTR
809                     LDA #$01               875                 CMP #$0D
810                     LDX #L,NAME2           876                 BEQ FEHLER3
811                     LDY #H,NAME2           877                 PLP
812                     JSR SETNAM             878                 PHP
813                     LDA #00                879                 BCC FEHLER2
```

```
880                JSR CHROUT           946                PHA
881                CLC                  947  PRB1          PLA
882                BCC FEHLER2          948                ASL A
883  FEHLER3       JSR CLRCHN           949                PHA
884                PLP                  950                SED
885                RTS                  951                LDA BUFFER+1
886                ;                    952                ADC BUFFER+1
887                ;                    953                STA BUFFER+1
888  FEHLER10      LDA #L,TEXT18        954                LDA BUFFER
889                LDY #H,TEXT18        955                ADC BUFFER
890                JSR TEXTAUS          956                STA BUFFER
891  FEHLER11      JSR TASTE            957                CLD
892                CMP #'1'             958                DEX
893                BNE FEHLER12         959                BNE PRB1
894                CLC                  960                PLA
895                RTS                  961                LDA BUFFER
896  FEHLER12      CMP #'2'             962                ORA #$30
897                BNE FEHLER11         963                STA BUFFER
898                SEC                  964                LDA BUFFER+1
899                RTS                  965                AND #$0F
900                ;                    966                ORA #$30
901                ;                    967                STA BUFFER+2
902  TAST          JSR GETIN            968                LDA BUFFER+1
903                BEQ TAST             969                LSR A
904                RTS                  970                LSR A
905                ;                    971                LSR A
906                ;                    972                LSR A
907  SPACE         LDA #$20             973                ORA #$30
908                .BY $2C              974                STA BUFFER+1
909  RETURN        LDA #$0D             975                LDA #$1D
910                JMP CHROUT           976                STA *$D3
911                ;                    977                LDX #$00
912                ;                    978  PRB2          LDA BUFFER,X
913  TASTE         JSR TAST             979                CMP #$30
914                CMP #'←'             980                BNE PRB3
915                BEQ TASTE1           981                LDA #$20
916                RTS                  982                STA BUFFER,X
917  TASTE1        JSR CLRCHN           983                INX
918                LDA DEV              984                CPX #$03
919                JSR CLOSE            985                BCC PRB2
920                LDA #$02             986  PRB3          LDX #$00
921                JSR CLOSE            987  PRB4          LDA BUFFER,X
922                LDA #$0F             988                JSR CHROUT
923                JSR CLOSE            989                INX
924                LDX #$F7             990                CPX #$03
925                TXS                  991                BCC PRB4
926                JMP START            992                RTS
927                ;                    993
928                ;                    994
929  TEXTAUS       STA *$57             995  DEVICE        LDA #$93
930                STY *$58             996                JSR CHROUT
931                LDY #$00             997                LDA #L,TEXT28
932  TXT1          LDA ($57),Y          998                LDY #H,TEXT28
933                BEQ TXT2             999                JSR TEXTAUS
934                JSR CHROUT           1000               LDA QUELL
935                INY                  1001               ORA #$30
936                BNE TXT1             1002               JSR CHROUT
937  TXT2          RTS                  1003               LDA #$9D
938                ;                    1004               JSR CHROUT
939                ;                    1005               JSR GETNUM
940  PRINTBLK      LDX #$00             1006               STA QUELL
941                STX BUFFER           1007               LDA #L,TEXT29
942                STX BUFFER+1         1008               LDY #H,TEXT29
943                LDX FILENR           1009               JSR TEXTAUS
944                LDA FILEL,X          1010               LDA ZIEL
945                LDX #$08             1011               ORA #$30
```

```
1012                JSR CHROUT
1013                LDA #$9D
1014                JSR CHROUT
1015                JSR GETNUM
1016                STA ZIEL
1017                LDA QUELL
1018                JSR DEVTEST
1019                BNE DEVOFF
1020                LDA ZIEL
1021                JSR DEVTEST
1022                BNE DEVOFF
1023                RTS
1024      DEVOFF    LDA #$02
1025                STA $D020
1026                LDA #L,TEXT31
1027                LDY #H,TEXT31
1028                JSR TEXTAUS
1029                JSR TASTE
1030                LDA #$0B
1031                STA $D020
1032                JMP DEVICE
1033
1034
1035      DEVTEST   LDY #$00
1036                STY *$90
1037                JSR LISTEN
1038                JSR UNLSN
1039                LDA *$90
1040                RTS
1041
1042
1043      GETNUM    LDY #$00
1044      GETNUM1   JSR CHRIN
1045                STA DEV,Y
1046                INY
1047                CMP #$0D
1048                BNE GETNUM1
1049      GETN0     LDA DEV
1050                CMP #$38
1051                BEQ GETN1
1052                CMP #$39
1053                BEQ GETN1
1054      GETN0.5   LDA #$14
1055                JSR CHROUT
1056                DEY
1057                BNE GETN0.5
1058                LDA #$38
1059                JSR CHROUT
1060                LDA #157
1061                JSR CHROUT
1062                JMP GETNUM
1063      GETN1     AND #$0F
1064                RTS
1065
1066
1067      DEVICE?   LDA QUELL
1068                CMP ZIEL
1069                BEQ DEV?4
1070      DEV?1     LDA #L,TEXT30
1071                LDY #H,TEXT30
1072                JSR TEXTAUS
1073      DEV?2     JSR TASTE
1074                CMP #$38
1075                BEQ DEV?3
1076                CMP #$39
1077                BNE DEV?2
```

```
1078    DEV?3     AND #$0F
1079    DEV?4     STA DEV
1080              RTS
1081
1082
1083    TEXT1     .BY $93 $0D $0D $20 $20 $20 $20
1084              .BY $20 $20
1085              .BY ' **** SUPER COPY 1541 ****' $0D
1086              .BY $20 $20 $20 $20 $20 $20
1087              .BY ' (C) BY N.MANN & D.WEINECK' $0D
1088              .BY $0D $0D $0D $20 $20 '1. DIRECTORY'
1089              .BY $0D
1090              .BY $20 $20 '2. KOPIEREN' $0D
1091              .BY $20 $20 '3. FORMATIEREN' $0D
1092              .BY $20 $20 '4. SCRATCH' $0D
1093              .BY $20 $20 '5. VALIDIEREN' $0D
1094              .BY $20 $20 '6. DEV.NR EINSTELLEN'
1095              .BY $0D $0D
1096              .BY $20 $20 '7. ENDE' $0D
1097              .BY $0D $0D $0D $20 $20 $12 $20
1098              .BY 'BITTE WAEHLEN SIE : ' $92 $20 $00
1099    TEXT2     .BY $0D $0D $20
1100              .BY 'FILE ZU LANG' $00
1101    TEXT3     .BY $0D $0D $20 $12
1102              .BY ' KOPIERLISTE VOLL ' $92 $00
1103    TEXT4     .BY 'JA/NEIN' $9D $9D $9D $9D $9D
1104              .BY $9D $9D $00
1105    TEXT5     .BY $12 $20 'JA' $20 $92 $20 $20
1106              .BY $20 $00
1107    TEXT6     .BY $0D $20 $20 $12
1108              .BY ' FALSCHER FILETYP ' $92
1109              .BY $20 $20 $20 $20 $20 $12 '↑↑↑' $92
1110              .BY $0D $00
1111    TEXT7     .BY $12 'NEIN' $92 $20 $20 $20
1112              .BY $00
1113    TEXT8     .BY $0D $0D $0D $0D $20
1114              .BY 'KOPIERVORGANG:' $0D $0D
1115              .BY $20 $20 '1. FORTLAUFEND' $0D
1116              .BY $20 $20 '2. EINZELN' $0D $0D $0D
1117              .BY $00
1118    TEXT9     .BY $0D $0D $0D $20
1119              .BY '1. DIRECTORY' $0D
1120              .BY $20 '2. VALIDIEREN' $0D
1121              .BY $20 '3. FORMATIEREN' $0D $00
1122    TEXT10    .BY $0D $0D $20 'VALIDIEREN ...'
1123              .BY $0D $00
1124    TEXT11    .BY $0D $0D $20 $12 ' QUELL-'
1125              .BY 'DISKETTE EINLEGEN ' $92 $0D $00
1126    TEXT12    .BY $0D $0D $20 'READING ' $00
1127    TEXT13    .BY $0D $0D $20 $12 ' ZIEL-'
1128              .BY 'DISKETTE EINLEGEN ' $92 $0D $00
1129    TEXT14    .BY $0D $0D $20 'WRITING ' $00
1130    TEXT15    .BY $0D $0D $20 $20 $12
1131              .BY ' K O P I E   F E R T I G ! '
1132              .BY $92 $0D $00
1133    TEXT16    .BY $0D $0D '  DISKNAMEN UND '
1134              .BY 'ID EINGEBEN' $0D $0D $00
1135    TEXT17    .BY $0D $0D $20 'NOCH EIN '
1136              .BY 'VERSUCH ?' $0D $00
1137    TEXT18    .BY $0D $0D $20
1138              .BY ' 1. UEBERSPRINGEN ?' $0D
1139              .BY $20 ' 2. NEUER VERSUCH ?' $0D $00
1140    TEXT19    .BY $0D $0D $20
1141              .BY 'NAECHSTES FILE :' $00
1142    TEXT21    .BY $0D $20 $20 $12 ' SCRATCH-'
1143              .BY 'DISK EINLEGEN ' $92 $0D $00
```

```
1144    TEXT22     .BY $0D $0D $20 $12 ' SICHER ? '
1145               .BY '- SPACE, SONST ← ' $92 $0D $00
1146    TEXT23     .BY $0D $0D ' SCRATCHING ' $00
1147    TEXT24     .BY ' BLOECKE ZU KOPIEREN ' $92
1148               .BY $0D $00
1149    TEXT25     .BY ' BLOECKE ZU SCRATCHEN ' $92
1150               .BY $0D $00
1151    TEXT26     .BY $0D ' *** SPACE ***' $0D $20
1152               .BY ' FUER WEITER' $0D $0D
1153               .BY $20 $12 $20 'BITTE WAEHLEN SIE '
1154               .BY $92 $0D $00
1155    TEXT27     .BY $20 '4. NOCHMAL KOPIEREN'
1156               .BY $0D $00
1157    TEXT28     .BY $0D $0D
1158               .BY 'QUELL-DISK NR: ' $00
1159    TEXT29     .BY $0D $0D
1160               .BY ' ZIEL-DISK NR: ' $00
1161    TEXT30     .BY $0D $0D $0D
1162               .BY 'GERAET NR. 8 ODER 9 ?' $00
1163    TEXT31     .BY $0D $0D $20 $20 $20 $20 $12
1164               .BY ' GERAET NICHT ANGESCHLOSSEN '
1165               .BY $92 $00
1166    NAME1      .BY '$' $00
1167    NAME2      .BY 'I' $00
1168    LESEN      .BY ',X,R' $00
1169    SCHREIB    .BY ',X,W' $00
1170               .EN
```

11.12.2 Schnell kopiert mit Hypra-Copy

Hypra-Copy ist ein schnelles und komfortables Filecopy-Programm für den C64. Das Kopieren wird um das Vier- bis Fünffache beschleunigt; somit ist die »Verträglichkeit« um einiges geringer als bei Super Copy (11.12.1). Dennoch ist das Programm insgesamt recht kurz, es belegt 15 Blöcke auf Diskette. So merkt sich Hypra-Load bis zu 30 Filenamen, wobei in einem Durchgang bis zu 232 Blöcke eingelesen werden. Das Programm wird über RUN gestartet.
Es erscheint dann das Hauptmenü:

```
HYPRA-COPY

:C: Copy Files
:S: Scratch Files
:D: Directory
:O: Order Disk
```

Die Bedienung ergibt sich damit eigentlich schon von selbst. Drückt man die Taste <O>, erscheint auf dem Bildschirm eine eckige Klammer mit dem blinkenden Cursor. Nun kann man den Befehl eingeben, der zur Floppy geschickt werden soll. Reagiert die Floppy mit einer Fehlermeldung, so wird diese auf dem Bildschirm ausgegeben.
Bei Betätigung von <D> erscheint das Directory. Der Ausdruck wird durch <CTRL> angehalten; durch <S> gelangt man in den Scratch-

Das linke Diskettensymbol auf dem Bildschirm steht für die Quelldiskette, das rechte repräsentiert die Zieldiskette. So sind Sie stets über den aktuellen Stand des Kopiervorgangs informiert.
Nun zur Bedienung. Master-Copy wird mit LOAD "MASTER-COPY V1.7" geladen und mit RUN gestartet. Es erscheint ein Menü auf dem Bildschirm, in dem Sie alle wichtigen Parameter und Steuerfunktionen einstellen können. Durch Drücken der Taste <B> aktivieren Sie den Menüpunkt zum Senden eines Befehls an die angeschlossene Floppystation. Sie können auf diese Weise Disketten formatieren, validieren, initialisieren und vieles mehr.
Drücken Sie auf <D>, erscheint das Directory der gerade eingelegten Diskette auf dem Bildschirm. Damit Ihnen keine Informationen verlorengehen, stoppt die Ausgabe automatisch, sobald der Bildschirm voll ist. Jetzt fährt der Computer erst auf Tastendruck mit der Anzeige fort. Die beiden Tasten <S> und <E> für »Starttrack« und »Endtrack« gestatten Ihnen die Einstellung des Bereiches einer Diskette, den Sie kopieren wollen. Das funktioniert von lediglich einer einzigen Spur bis hin zu 40spurig formatierten Disketten.
Wollen Sie nicht auf einem Laufwerk der Nummer 8, sondern vielmehr auf einer Floppystation mit der Nummer 9, 10 oder 11 kopieren, so ist auch das kein Problem. Ein Druck auf <G>, und schon können Sie zwischen vier verschiedenen Geräteadressen umschalten.
Mit <K> starten Sie den Kopiervorgang. Die Bildschirmanzeige wechselt jetzt auf die Statusinformationen für das Duplizieren von Disketten, und der Computer fordert zum Einlegen der Quelldiskette auf. Diese Aufforderung ist nicht etwa in Worte gefaßt, sondern mit Hilfe des Diskettensymbols realisiert. Ist das linke Symbol auf dem Bildschirm dunkel gefärbt, so heißt das: Einlegen der Quelldiskette. Ein schattiertes rechtes Symbol sagt Ihnen: Bitte legen Sie nun die Zieldiskette ein. In der Mitte zwischen beiden Zeichen sehen Sie drei Anzeigen, wobei die laufende Nummer des Diskettenwechsels, eventuell aufgetretene Fehler und die aktuelle Kopierzeit dargestellt werden.
Tritt beim Lesen oder Schreiben ein Fehler auf, so wird der Kopiervorgang stark abgebremst, da Master-Copy mehrere Schreib- und Leseversuche unternimmt. Läßt sich ein Fehler nicht beseitigen, so erscheint dessen Meldung im Klartext auf dem Bildschirm. Ein Reparieren von defekten Sektoren erfolgt nicht; Master-Copy überträgt im Fall eines Defekts einen Sektor mit Leerinhalt auf die Zieldiskette.

11.13.2 Kopieren ohne Grenzen: Copy +

Für Besitzer eines Parallelkabels, wie es die gängigen hardwaremäßigen Floppy-Speeder haben, bietet »Copy+« im Zusammenhang mit einer Floppy 1541 zwei große Vorteile:

1. Auch kopiergeschützte Originale lassen sich damit in weiten Grenzen kopieren, um Sicherheitskopien (!) zu erstellen.
2. Die Übertragung defekter Datenblöcke ist mit »Copy+« möglich, die Reparatur nicht.

Für normale Disketten ist hingegen »Master-Copy V1.7« (11.13.1) empfehlenswerter. Auch die Voraussetzung eines Parallelkabels zwischen Floppy und Computer wird nicht von allen C64-Anwendern erfüllt. Außerdem muß man zur effektiven Bedienung von »Copy+« eine gehörige Portion fundierten Floppy-Know-hows mitbringen.
»Copy +« wird mit RUN gestartet und bietet folgendes Hauptmenü an:

<A>endern der Speedflags
<J>ustierung aendern
<C>opiere Disk
<T>rackcheck
<S>ectorshorting on/off

<A>: Ändern der Speedflags

Mit Hilfe der Cursortasten und der Tasten <0> bis <3> kann die Geschwindigkeit jeder Spur beim Lesen und Schreiben geändert werden. »Copy +« liefert hier eine Voreinstellung, die von Spur 1 bis 35 dem normalen Aufzeichnungsformat entspricht. Jede Spur wird mit einer bestimmten Geschwindigkeit aufgezeichnet, die schon beim Formatieren festgelegt ist. Wie bei einer Schallplatte ist auf den äußeren Spuren die Geschwindigkeit höher als auf den inneren. Es stehen hier vier Geschwindigkeiten zur Verfügung, die schon beim Formatieren der Diskette eingestellt sind:

Spur 1 bis 17 Speed 3
Spur 18 bis 24 Speed 2
Spur 25 bis 30 Speed 1
Spur 31 bis 35 Speed 0
Spur 36 bis 40 Speed 2

Nun ist die Floppy 1541 auch in der Lage, 40 Spuren zu beschreiben und zu lesen. Copy + kann selbstverständlich auch diese 40 Spuren kopieren und bietet als Voreinstellung die Geschwindigkeit 2 für die Spuren 36–40 an. Dies kann aber von Diskette zu Diskette, falls 40spurig formatiert, verschieden sein. Zur Ermittlung der Geschwindigkeit nimmt man den Menüpunkt <T>.

<C>: Copy Disk

Geben Sie zuerst Anfangs- und Endttrack an (voreingestellt: 1 und 40). Mit <RETURN> beginnt der Kopiervorgang. Während des Diskettenzugriffes ist jeweils der Bildschirm abgeschaltet, bei Aufforderung zum Diskettenwechsel erscheint eine Statusmeldung. Diese umfaßt (von links nach rechts) die Anzahl der gelesenen Bytes, die Anzahl der geschriebenen Bytes sowie die Position eines eventuellen Verify-Errors. Für diesen letzten Fehler gibt es drei Gründe:

1. Positionsnummer niedrig: Zieldiskette schreibgeschützt. (Aufkleber entfernen!)
2. Track-Kapazität wegen falscher Geschwindigkeit oder vergessenem »Sectorshorting« nicht ausreichend.
3. Defekte Zieldiskette.

Sie haben dabei die Wahl zwischen <T> zum Wiederholen (Try again), <S> zum Überspringen (Skip) und <Q> zum Rücksprung ins Hauptmenü (Quit).

<J>: Justierung

Bei gestartetem Kopiervorgang sucht Copy + zuerst Spur 18, wo sich die BAM befindet. Sollte dieser Track auf Anhieb nicht gefunden werden, so fährt der Schreib-Lese-Kopf bis zum Anschlag (Spur 1) und dann wieder bis Spur 18. Nun können Sie Copy + auch dazu veranlassen, Halbspuren zu kopieren. Abhängig von der Eingabe bei <X> wird der Kopf, nachdem er Spur 18 gefunden hat, um die gewünschte Anzahl von Halbspuren nach innen oder außen verschoben. Die Eingabe einer negativen Zahl bewirkt eine Verschiebung in Richtung Spur 1. Zum Kopieren von 34,5 ist beispielsweise –1 einzugeben und dann Spur 35 zu duplizieren. Es wird die jeweils bei <A> voreingestellte Geschwindigkeit zum Kopieren genutzt.

<S>: Sectorshorting

Es erscheint eine Eingabezeile, in der die Sektorlänge (Zahl der Bytes zwischen zwei Sync-Markierungen) wählbar ist. Voreingestellt sind 328 Byte. Die physikalische Länge eines Blockes umfaßt nämlich mehr als die 256 Byte, in denen die Daten gespeichert sind. Zu jedem Block gehört eine Sync-Markierung, der Blockheader mit Prüfsumme, dann der eigentliche Datenblock und zuletzt eine Prüfsumme über den Datenblock. Diese Blocklänge kann durch gewisse Maßnahmen manipuliert werden. Sollten beim Kopieren Fehler auftreten, so sollte man die Sektorlänge größer wählen. Die nächste Sync-Markierung wird automatisch erkannt.

<T>: Trackcheck

Der hier eingegebene Track wird mit der bei <A> eingestellten Geschwindigkeit gelesen. Sollte dies nicht möglich sein, gibt Copy + eine Fehlermeldung aus, und die Geschwindigkeit ist neu einzustellen.
Der erste eingelesene Track steht ab $2000 im Speicher des C64; ab $C000 ist noch Platz für einen Monitor.

12 Aktives Anwenden oder: Kniffe zur Software

Die Softwarevielfalt für den C64 richtig einzusetzen, erfordert mehr als bloß Disketten voller Programme. Richtig umgehen kann man erst mit den richtigen Tricks, die aus jeder Software noch mehr herausholen, als man auf den ersten Blick meint.

12.1 Spiele-Pokes oder: So kann's jeder

Sie kennen die Situation: Da weiß man beim Spielen, was für tolle Level (Spielstufen) noch auf einen warten, bringt aber nicht die nötige Geschicklichkeit mit, bis in die neuen Ebenen vorzustoßen. Aufgeben? Nicht, wenn Sie einen geheimnisvollen Befehl eingeben, das Spiel wie gewohnt starten, und siehe da: Die Zahl der Spielfiguren hat sich vervielfacht, oder der »Held« ist immun gegen jeden Angriff.

12.1.1 POKE-Liste

Im folgenden erhalten Sie eine Aufstellung von POKE-Befehlen zu bekannten Spielen; diese haben Sie nach dem Laden des Programms, aber vor dem Start allesamt einzugeben. Für das Funktionieren kann nicht garantiert werden; vor allem Autostart-Programmen ist mit diesen POKEs leider nicht beizukommen, weil man keine Gelegenheit zur Eingabe hat (vor dem Laden bringt es nichts, nach dem Laden erfolgt eigenständig ein Start). Auch Modulversionen wehren sich hartnäckig gegen POKEs.
Wenn jedoch aufgrund der POKEs merkwürdige Nebeneffekte auftreten (wirre Grafikzeichen anstelle »vernünftiger« Spielfigurenzahlen o.ä.), so ist dies oftmals durchaus in Ordnung.
Damit Sie nun nicht nur staunen, wie die Befehle funktionieren, sei ein Beispiel auf Basic-Ebene gegeben. Nehmen wir an, in einem Spiel finden wir folgende Zeile:

```
1590 IF CRASH=1 THEN LEBEN=LEBEN-1:PRINT "SIE HABEN NOCH "LEBEN"
     VERSUCHE":GOTO 7845
```

So würde man nach folgender Änderung wahrscheinlich eine »Trainerversion« haben:

```
1590 IF CRASH=1 THEN PRINT "SIE HABEN NACH WIE VOR "LEBEN" VERSUCHE"
     :GOTO 7845
```

Das Verringern der Zahl LEBEN wird einfach unterdrückt. Und die POKE-Befehle, die Sie hier erfahren, bewirken adäquate Änderungen; nur sind die guten Spiele eben in Maschinensprache geschrieben, und da gibt es für Außenstehende keine anderen Zugriffsmöglichkeiten als über POKE.
Hier nun die POKEs:

Bagitman:	POKE19013,189:POKE22236,255
Battle through time:	POKE17861,12:POKE17864,12
Battle Zone:	POKE8909,100
Black Hawk:	POKE8290,255
Blagger:	POKE3560,8
Bruce Lee:	POKE5686,128:POKE5672,128
Burnin' Rubber:	POKE18432,173
Buck Rogers:	POKE8825,36
China Miner:	POKE34623,44
Choplifter:	POKE8011,173
Crazy Kong:	POKE30624,173
Crossfire:	POKE27625,173
Dig Dug:	POKE10473,255
Dinkey Doo:	POKE12296,165 und/oder POKE11989,18
Fort Apocalypse:	POKE1496,0:POKE14760,0:POKE36366,0
Frantic Freddy:	POKE34535,24
Frogger:	POKE22341,173
Galaga:	POKE17388,173
Galaxians:	POKE7065,230:POKE17288,165
Gangster:	POKE5989,58
Hard Hat Mack:	POKE16877,173
Hunchback:	POKE9521,44
Jet Set Willy:	POKE11345,33
Jumping Jack:	POKE27904,173
Jumpman Junior:	POKE9450,44
Jungle Hunt:	POKE2242,165
Lady Tut:	POKE2392,50
Laser Strike:	POKE16475,173
Lazy Jones:	POKE40606,255
Moon Buggy:	POKE24151,173
Mr. Robot:	POKE11518,255
Neptune:	POKE 7870,60
Pitfall:	POKE5393,255
Pogo Joe:	POKE2779,36

Pooyan:	POKE20634,173
Quest For Tires:	POKE7341,99:POKE11485,125:POKE14864,0
R-Nest:	POKE4446,173
Revenge of M.Camels:	POKE40905,4:POKE40905,100
sowie:	POKE 35518,250
Sammy Lightfoot:	POKE3678,189
Seafox:	POKE7337,173
Shamus:	POKE18486,169:POKE23558,169
Shamus Case II:	POKE15476,176
Snokie:	POKE33242,255
Super Scramble:	POKE4691, <Tempo>
Survivor:	POKE19563,255
Zeppelin:	POKE18546,44

Trick für Miner 2049'er: Halten Sie jeweils die Leertaste oder wahlweise den Joystick-Feuerknopf konstant gedrückt. Dies führt sie nach einiger Zeit »von selbst« ins nächste Bild, um die höheren Levels betrachten zu können.
Es folgen drei Programme, die ebenfalls zum Bezwingen anspruchsvoller Spiele dienen sollen.

12.2 Spritekill oder: Schach den Sprites

Mit »Spritekill« können Sie jeden unerwünschten Schuß, jeden noch so gemeinen Gegner auslöschen. Kenntnisse über den C64-Speicheraufbau sind wünschenswert, ja fast sogar notwendig.

Listing 12.1: Spritekill

```
0 REM *****************
1 REM * SPRITE - KILL *
2 REM *****************
3 REM *  V O N
        *
4 REM *****************
5 REM *  G. GLENDOWN  *
6 REM *****************
7

10 PRINT"(CLR)(WHT)SPRITEKILL"
   POKE53281,0
   POKE53280,0
   PRINT"VON GARRY GLENDOWN"
20 INPUT"STARTADRESSE";SA$
   SA=VAL(SA$)
21 IFSA>40429ANDSA<49152ORSA>52716ORSA<1024THENPRINT"UNGUELTIG!"
   GOTO20
29 PRINT"ENDADRESSE
   "SA+532
   INPUT"OK? J/N";A$
   IFA$<>"J"THEN20
30 IFSA<6000THENFL=1
35 IFFL=1THEN100
```

```
40 INPUT"ABSPEICHERN J/N";A$
   IFA$="J"THENFL=1
50 FORT=SATOSA+531
   READA
   POKET,A
   CH=CH+A
   NEXT
   IFCH<>65921THENPRINT"FEHLER!"CH
   END
51 CH=0
60 READCO
   CH=CH+CO
   IFCO=-1THEN200
70 B=PEEK(SA+CO)+256*PEEK(SA+CO+1)
   B=B-28672+SA
   H=INT(B/256)
   L=B-H*256
80 POKESA+CO,L
   POKESA+CO+1,H
   GOTO60
100 FORT=49152TO49683
    READA
    POKET,A
    CH=CH+A
    NEXT
    IFCH<>65921THENPRINT"FEHLER!"CH
    END
101 CH=0
110 READCO
    CH=CH+CO
    IFCO=-1THEN140
120 B=PEEK(49152+CO)+256*PEEK(49153+CO)
    B=B-28672+SA
    H=INT(B/256)
    L=B-H*256
130 POKE49152+CO,L
    POKE49153+CO,H
    GOTO110
140 IFCH<>22866THENPRINT"FEHLER!"
    END
141 INPUT"FILENAME";F$
    OPEN1,8,2,F$+",P,W"
150 H=INT(SA/256)
    L=SA-H*256
    PRINT#1,CHR$(L)CHR$(H);
160 FORT=49152TO49683
    PRINT#1,CHR$(PEEK(T));
    NEXT
    CLOSE1
    OPEN1,8,15
170 INPUT#1,A,B$,C,D
    PRINTA","B$","C","D
    END
200 IFCH<>22866THENPRINT"FEHLER!"CH
    END
201 IFFL=1THEN300
210 PRINT"PROGRAMM IM SPEICHER VON"SA" BIS"SA+531
220 PRINT"PROGRAMMSTART MIT (RVS) SYS";SA" (OFF)"
230 END
300 H=INT(SA/256)
    L=SA-H*256
310 INPUT"FILENAME ";F$
    OPEN1,8,2,F$+",P,W"
    PRINT#1,CHR$(L)CHR$(H);
```

```
320 FORT=SATOSA+531
    PRINT#1,CHR$(PEEK(T));
    NEXT
    CLOSE1
    OPEN1,8,15
    INPUT#1,A,B$,C,D
    CLOSE1
330 PRINTA",","B$",","C",","D
    IFA=0THEN210
340 INPUT"NOCHMAL VERSUCHEN J/N";A$
    IFA$<>"J"THEN210
350 OPEN1,8,15,"S
    "+F$
    CLOSE
    GOTO300
32000 DATA120,169,3,162,0,134,255,134,248,141,21,208,141,23,208,141,29,208,169
32001 DATA100,141,0,208,141,3,208,162,200,142,2,208,206,40,208,141,1,208,162
32002 DATA13,142,248,7,232,142,249,7,32,60,113,32,133,112,169,19,32,210,255,165
32003 DATA255,32,102,112,165,254,32,102,112,169,32,32,210,255,32,133,112,165
32004 DATA253,32,102,112,165,252,32,102,112,32,164,112,76,169,113,169,19,32,210
32005 DATA255,169,36,32,210,255,96,72,74,74,74,74,201,10,48,2,105,6,105,48,32
32006 DATA210,255,104,41,15,201,10,48,2,105,6,105,48,32,210,255,96,165,254,41
32007 DATA3,170,189,16,114,24,101,248,133,252,165,254,74,74,133,253,165,255,170
32008 DATA189,16,114,24,101,253,133,253,96,32,133,112,160,63,120,169,53,133,1
32009 DATA177,252,145,249,136,16,249,169,55,133,1,88,96,76,129,255,165,254,24
32010 DATA105,16,133,254,176,67,76,49,112,165,254,56,233,16,133,254,144,73,76
32011 DATA49,112,230,254,240,48,76,49,112,165,254,56,233,1,133,254,144,54,76
32012 DATA49,112.173,28,208,73,3,141,28,208,76,49,112,173,29,208,73,3,141,29
32013 DATA208,76,49,112,173,23,208,73,3,141,23,208,76,49,112,230,255,165,255
32014 DATA201,4,16,3,76,49,112,169,0,133,255,76,49,112,198,255,48,3,76,49,112
32015 DATA169,3,133,255,76,49,112,32,133,112,32,78,113,160,63,169,0,136,145,252
32016 DATA208,251,76,214,112,162,64,160,3,134,249,132,250,96,162,128,160,3,134
32017 DATA249,132,250,96,32,69,113,32,164,112,32,60,113,96,32,133,112,32,69,113
32018 DATA169,249,162,252,141,175,112,142,177,112,32,164,112,32,133,112,169,252
32019 DATA162,249,141,175,112,142,177,112,32,60,113,76,49,112,230,248,165,248
32020 DATA201,63,208,4,169,0,133,248,76,49,112,198,248,165,248,201,255,208,4
32021 DATA169,62,133,248,76,49,112,169,0,133,248,76,49,112,32,78,113,76,49,112
32022 DATA160,0,162,208,165,203,200,208,6,232,208,3,76,188,113;197,251,240,241
32023 DATA133,251,160,15,185,226,113,197,251,240,6,136,16,246,76,169,113,136
32024 DATA152,10,168,185,242,113,141,222,113,185,243,113,141,223,113,76,221,113


32025 DATA76,49,112,40,43,54,57,36,23,25,28,60,3,2,7,17,63,20,214,112,221,112
32026 DATA190,112,202,112,233,112,244,112,255,112,10,113,42,113,156,113,126,113
32027 DATA141,113,88,113,187,112,163,113,0,64,128,192
60000 DATA47,50,60,65,73,78,83,86,89,139,156,165,200,212,219,231,242,253,264
60010 DATA275,282,289,296,299,302,314,335,338,341,345,348,355,358,361,364
60020 DATA371,374,377,380,395,410,417,420,423,438,449,459,466,469,472,475,478
60030 DATA498,500,502,504,506,508,510,512,514,516,518,520,522,524,526,-1
```

»Spritekill« hilft, in einem Spielprogramm alle unliebsamen Sprites zu entfernen. Bonusgegenstände und eigene Figur bleiben selbstverständlich im Spiel – vielleicht noch ein paar einfache Gegner, damit es nicht zu langweilig wird. Der DATA-Lader aus Listing 12.1 legt Spritekill in beliebige Speicherbereiche, wofür sich meistens der Bildschirmspeicher anbietet. Bei Spielen mit bis zu 186 Blöcken gibt man einfach »49152« an. Von dieser eingegebenen Adresse hängt die spätere Aktivierung ab; im Beispiel wäre nach Ablauf von Spritekill also »SYS 49152« der entsprechende Startbefehl.

Alle Funktionen werden mit insgesamt 15 Tasten gesteuert:

<+>	nächster Spriteblock
<–>	vorheriger Spriteblock
<↑>	16 Sprites vor
<←>	16 Sprites zurück
<B>	Weiterschalten des 16-Kbyte-Bereiches
<M>	Multicolorgrafik ein
<X>, <Y>	Umschaltung der Sprite-Ausdehnung
<Leertaste>	Löschen des linken Bereichs
<C>	Kopieren des linken Sprites in den Puffer
<R>	Kopieren des rechten Sprites in den Puffer
<CRSR RIGHT>	Erhöhen der Startadr. des linken Sprite
<CRSR DOWN>	Erniedrigen der Startadr. des linken Sprite
<F7>	Zurücksetzen der Einzelbyte-Verschiebungen
<RUN/STOP>	Verlassen des Programms

Um mit »Spritekill« möglichst effektiv zu arbeiten, bietet sich folgende Strategie an:

- Nach dem Starten von »Spritekill« durchsucht man den Speicher zunächst in größeren Schritten mit den beiden Pfeiltasten.
- Sobald man ein Sprite erkennt, geht man mit <+> und <–> so lange weiter, bis das letzte Sprite kommt.
- Ab dieser Stelle durchsucht man den Speicher in der anderen Richtung und löscht die ungewollten Sprites einfach mit der <Leertaste>.
- Zum Schluß drückt man <RUN/STOP> und speichert bzw. startet das modifizierte Spiel.

12.3 Adventurelister

Kommt man bei Abenteuerspielen (Adventures) nicht mehr weiter, bietet sich eine Untersuchung der Texte an. Aber Einsteiger können mit Maschinensprache-Monitoren nicht recht umgehen. Daher wurde ein Basic-Programm entwickelt, das die Texte in Adventures sucht und auf dem Bildschirm ausgibt.

Listing 12.2: Adventurelister

```
10 X=2048
20 Y=PEEK(X)
   IFY>64ANDY<91THENPRINTCHR$(Y);
   GOTO40
30 PRINT".";
40 X=X+1
   GETR$
   IFR$<>""THEN60
50 GOTO20
60 IFR$="Q"THENEND
70 GETY$
   IFY$=""THEN60
80 GOTO20
```

So wird das Programm gehandhabt:

1. Adventure laden
2. Reset-Knopf drücken
3. Programm »ADVENTURE-LISTER« laden und starten

Eine Programmunterbrechung erreicht man durch Drücken einer beliebigen Taste; weiter geht es bei erneutem Druck auf eine Taste. Einen Programmabbruch erzielt man mit <Q>.

12.4 Spindizzy-Trainer

Eines der Spiele, die mich am meisten faszinierten, ist »Spindizzy«, das Murmelspiel mit der schier endlosen Anzahl neuer Level. Deshalb möchte ich Ihnen den »Spindizzy Trainer« (Filename: »SPINDIZZY TRAIN.«) von Paul Shirley, dem Programmierer höchstpersönlich, nicht vorenthalten. Laden Sie diesen, legen Sie die Originaldiskette ein (Raubkopien werden nicht akzeptiert!) und starten Sie das Programm mit RUN. Spindizzy wird geladen, nur die Zeit läuft nicht mehr ab und Sie können in Ruhe spielen.

13 RAM-Disk-Steuerung oder: Der C576

Was lange nicht für möglich gehalten wurde, ist jetzt von GEOS bewiesen worden: Der C64 kann eine Speichererweiterung von zusätzlich 128 oder 512 Kbyte adressieren! Zusammen mit den 64 Kbyte der Grundeinheit ergäbe dies ein »576-Kbyte-RAM-SYSTEM«.
Die Erweiterungen 1700 (128 Kbyte zusätzlich) und 1750 (512 Kbyte zusätzlich) wurden ursprünglich für den C128 entwickelt, sind aber auch in den C64 einzustecken. Aufgrund des geringen Preisunterschieds zwischen beiden Erweiterungen ist der 1750 eindeutig der Vorzug zu geben, nicht zuletzt wegen der besseren Verwendbarkeit unter GEOS. Deshalb erfahren die interessierten Profis hier, wie im Prinzip mit der RAM-Erweiterung von Programmen aus umzugehen ist. Diese Informationen schlagen sich hoffentlich in interessanten Entwicklungen nieder!

13.1 RAM ist nicht gleich RAM

Wenn man die Erweiterung 1700 oder 1750 in den Expansion-Port einsteckt, tut sich zunächst nichts. Im Computerspeicher blenden sich jedoch ab $DF00 die Register ein, mit deren Hilfe die RAM-Disk angesprochen wird. Da der Adreßbus des C64 nach wie vor auf 16 Bit beschränkt bleibt, ist die CPU auch gar nicht in der Lage, den zusätzlichen Speicher wie gewohnt anzusprechen; dies geschieht nur auf dem Umweg über die DMA-Register und ist somit langsamer und aufwendiger als Zugriffe auf den Grundspeicher. Folgende Schritte sind beispielsweise beim Zugriff auf eine Adresse im zusätzlichen RAM durchzuführen:

1. Übergabe der gewünschten Adresse und des Speicherabschnittes an DMA-Register
2. Senden eines »Lese-Kommandos«
3. Auslesen eines Kommunikationsregisters, das den bereitgestellten Wert enthält

Da dies sehr aufwendig ist, wird zweckmäßigerweise nie ein einzelnes Byte gelesen, sondern immer nur ein Speicherabschnitt. Dabei findet im wesentlichen eine Übertragung von Speicherinhalten zwischen normalem C64-Speicher und Erweiterungsbereich statt. Folgende drei Möglichkeiten sind vorgesehen:

- STASH speichert Daten aus dem C64-Hauptspeicher im RAM-Erweiterungsspeicher
- FETCH überträgt Daten aus dem RAM-Erweiterungsspeicher in den C64-Hauptspeicher
- SWAP vertauscht Daten aus RAM-Erweiterung mit Daten aus dem C64-Hauptspeicher

Dazu sind jeweils bestimmte Parameter wichtig, die die DMA-Register geschrieben werden:

BYTES	gibt an, wie viele Bytes übertragen werden sollen
C64ADR	ist die Startadresse der Datenübertragung im C64-Hauptspeicher (0–65535)
RAMADR	ist die Zieladresse der Datenübertragung im RAM-Modul (0–65535)
RAMBNK	ist die Nummer der gewünschten RAM-Erweiterungsbank (bei 1700: 0 oder 1; bei 1750: 0 bis 7)

Die Parameter sind somit in 2 Byte darstellbar (Ausnahme: BANK belegt nur 1 Byte). Durch die Verwendung der Befehlscodes werden Daten aus einem RAM-Bereich in einen anderen übertragen. Befinden sich im gewählten Bereich Programme oder irgendwelche anderen Daten, werden diese automatisch überschrieben beziehungsweise bei SWAP vertauscht.

13.2 Die Register für DMA

Da ein 8-Bit-Prozessor wie der 6510 des C64 nur 65536 verschiedene Speicheradressen ansprechen kann, muß ein größerer Speicherbereich in Speicherbänke eingeteilt werden. Bild 13.1 zeigt diese Einteilung für die Erweiterung 1700, Bild 13.2 (Seite 433) für die 1750.
Die entsprechenden Routinen bedienen sich eines DMA; DMA bedeutet »Direct Memory Access«, was man mit »direkter Speicherzugriff« übersetzt. Ein DMA kann auf verschiedene Arten herbeigeführt werden, die allesamt im Handbuch der Erweiterungsmodule Erklärung finden. Tabelle 13.1 gibt Auskunft über die Adressierung des I/O-Bereiches der für DMAs wichtigen Register.

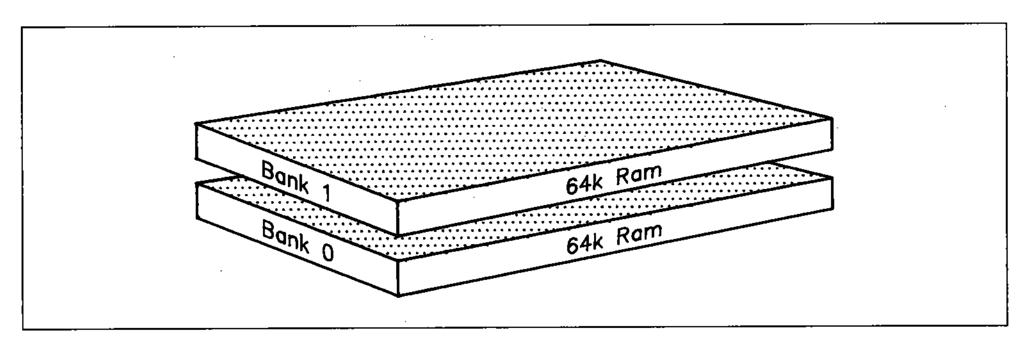


Bild 13.1: Die Organisation der RAM-Erweiterung 1700

Adresse	Bits	Funktion
$df00	7–0	Bit 4: Modulgröße (0 = 128K; 1 = 512K)
$df01	7–0	Kommando-Register für STASH: %10000100 FETCH: %10000101 SWAP: %10000110
$df02	7–0	Startadresse im C 64 (Low-Byte)
$df03	7–0	Startadresse im C 64 (High-Byte)
$df04	7–0	Startadresse in RAM-Erweiterung (Low-Byte)
$df05	7–0	Startadresse in RAM-Erweiterung (High-Byte)
$df06	7–0	Nummer der RAM-Erweiterungsbank (bei 1700: 0–1; bei 1750: 0–7)
$df07	7–0	Anzahl der zu übertragenden Bytes (Low-Byte)
$df08	7–0	Anzahl der zu übertragenden Bytes (High-Byte)

Tabelle 13.1: Die wichtigsten I/O-Register für DMAs

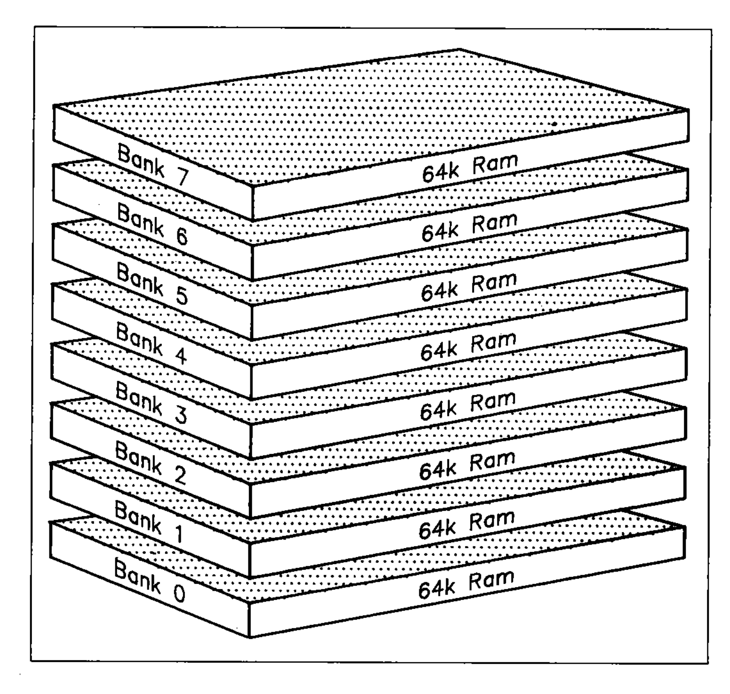


Bild 13.2. Die Organisation der RAM-Erweiterung 1750

Die Tabelle bezieht sich auf Daten des 1700/1750-Originalhandbuches (Seite 20). Wie ein DMA letztendlich gestartet wird, ist dem Listing 13.1 zu entnehmen (liegt nicht auf Diskette bei, da es in diesem Zustand nicht assemblierfähig ist).

Listing 13.1: So funktionieren typische DMA-Zugriffe

```
0 :
1 ;
2 ;              dokumentiertes quell-code listing zu
3 ;
4 ; dma-rel. (relokatibel = kann ohne adressen-umrechnung verschoben
5 ;                          werden)
6 ;
9 :
10 * = 51200
10000 ;
10001 ;flaggen definieren
10002 ;
10003 bankflg = $df06
10004 flgge   = 2
10005 ;
10006 ;durch variablen einsprung --> uebertragungs-art waehlen
10007 ;naehere beschreibung der befehle siehe anleitung !
10008 ;
10009 stash lda #%10000100   ;uebertragungsart waehlen
10010 .byt $2c               ;skip (nachfolgenden befehl ueberspringen)
10011 fetch lda #%10000101   ;-"-
10012 .byt $2c               ;skip
10013 swap  lda #%10000110   ;-"-
10014 ;
10015 pha                    ;nummer des befehls merken
10016 ;
10017 ;
10018 ;test auf ram-disk-version
10019 ;
10020 chdisk lda #0          ;status-register
10021        sta $df00       ;loeschen
10022        lda $df00       ;und relevantes bit testen
10023        and #16         ;
10024        bne skiper      ;
10025        lda #2          ;1700 (128k, 2 baenke)
10026 .byt $2c               ;skip
10027 skiper lda #8          ;1750 (512k, 8 baenke)
10028        sta bankflg     ;maximale bank-anzahl merken
10029 ;
10030 ;uebertragungswerte holen
10031 ;
10032 werget jsr $aefd       ;teste auf komma
10033        jsr $ad8a       ;hole anzahl bytes
10034        jsr $b7f7       ;konvertiere in adressformat
10035        lda $14         ;setze anzahl der zu
10036        sta $df07       ;uebertragenden bytes
10037        lda $15         ;in i/o register
10038        sta $df08       ;-"-
10039 ;
10040        jsr $aefd       ;teste auf komma
10041        jsr $ad8a       ;hole startadresse im c64
10042        jsr $b7f7       ;konvertiere in adress-format
10043        lda $14         ;setze startadresse
10044        sta $df02       ;in i/o register
10045        lda $15         ;-"-
10046        sta $df03       ;-"-
10047 ;
10048        jsr $aefd       ;teste auf komma
10049        jsr $b7eb       ;hole startadresse und banknummer
10050        lda $14         ;der ram-erweiterung
10051        sta $df04       ;setze in i/o register
10052        lda $15         ;-"-
10053        sta $df05       ;-"-
10054 ;
```

```
10055 cont    cpx bankflg        ;pruefe, ob banknummer mit der
10056         bcc skips2         ;tatsaechlichen zahl uebereinstimmt
10057 ;
10058         pla                ;nein, dann fehler
10059         ldx #16            ;"out of memory ..."
10060         jmp $e38b          ;-"-
10061 ;
10062 skips2 stx $df06           ;ok, dann in i/o register
10063         jsr $aefd          ;teste auf komma
10064         jsr $b79e          ;bank-option ins x-register
10065         cpx #2             ;falsche bank-option "?"
10066         bcc skup           ;nein, flagge setzen
10067         ldx #1             ;ja, flagge auf standard
10068         stx flgge          ;-"-
10069         pla                ;-"-
10070         ldx #14            ;und fehler
10071         jmp $e38b          ;"illegal quantity ..."
10072 ;
10073 skup    stx flgge          ;flagge setzen
10074         pla                ;nummer des befehls zurueckholen
10075         tay                ;und ins y-register
10076         lda flgge          ;flagge testen
10077         bne rom            ;rom-version
10078 ;
10079 ram     sty $df01          ;ram-version, befehlsnummer setzen
10080         sei                ;interrupt sperren
10081         lda $01            ;alte speicher-konfiguration
10082         pha                ;merken
10083         lda #0             ;alles auf ram
10084         sta $01            ;schalten
10085         sta $ff00          ;dma mit 'dummy' starten
10086         pla                ;alte konfiguration zurueckholen
10087         sta $01            ;und setzen
10088         lda #1             ;bank-option auf
10089         sta flgge          ;standard
10090         cli                ;interrupt wieder zulassen
10091 rts                        ;zurueck ins basic
10092 ;
10093 rom     lda #0             ;rom-version
10094         sty $df01          ;befehls-nummer setzen
10095         sta $ff00          ;und mit 'dummy' starten
10096 rts                        ;zurueck ins basic
10097 ;
10098 ;
10099 .asc "(c) dma-version" ;copyright-vermerk
10100 .asc " 1987 by t.pohl"
10101 .byt 0,0,0
10102 .end
```

14 Der Extended Color Mode

Haben Sie sich schon gefragt, warum sich denn nicht jedem Zeichen am Bildschirm neben seiner Schriftfarbe eine eigene Hintergrundfarbe zuordnen läßt? Hier ist eine einfache, aber äußerst wirkungsvolle Methode, die den Einsteiger verblüfft und selbst Insidern noch nicht unbedingt geläufig ist. Damit wird das Phänomen »Extended Color Mode« allen C 64-Besitzern zugänglich.

Programmierung

Listing 14.1 zeigt, wie man bis zu vier verschiedene Hintergrundfarben auf einmal darstellt:

Listing 14.1: BSP/EXT.COL.MODE

```
100 rem ********************************
110 rem *                              *
120 rem *  beispiel fuer den e.c.m.    *
130 rem *                              *
140 rem *    extended color mode       *
150 rem *  - mehrfarbiger textmodus    *
160 rem *                              *
170 rem ********************************
180 rem *                              *
190 rem * written 16.10.1987 by        *
200 rem *                              *
210 rem *      florian mueller         *
220 rem *                              *
230 rem ********************************
240

250

260 poke53265,peek(53265)or64
    rem *** extended color mode wird aktiviert
270 poke53281,5
     rem hintergrund #0 (normal) - 5 (gruen)
280 poke53282,7
     rem hintergrund #1 (shift)  - 7 (gelb)
290 poke53283,15
    rem hintergrund #2 (revers) - 15 (hellgrau)
300 poke53284,6
     rem hintergrund #3 (shf/rvs)- 6 (blau)
310 print"(clr)(rvs)(blk) demonstration fuer extended color mode (down)(down)(do
wn)(down)"
320 print"(blk)schwarz auf gruen(down)"
330 print"(red)ROT AUF GELB                            "
340 print"(blu)(rvs)dunkelblau auf hellgrau                 "
350 print"(wht)(rvs)WEISS AUF DUNKELBLAU                       "
```

Zeile 260: Durch Setzen von Bit 6 in Adresse 53265 (VIC-Register #17) aktiviert man den Extended Color Mode (erweiterter Farbmodus). Das Ausschalten geschieht durch POKE53265,PEEK(53265)AND191.
Im ECM werden nur die ersten 64 verschiedenen Zeichen dargestellt. Dafür stehen vier Hintergrundfarben zur Auswahl (macht ingesamt wieder 4*64 = 256 Zeichencodes, damit letztlich alles beim alten bleibt).

Zeilen 270–300: Die vier Hintergrundfarben werden als Farbcodes (0–15) in den Adressen 53281–53284 untergebracht. 53281 ist, Sie wissen das sicher, das Register für die »herkömmliche« Hintergrundfarbe. Die Farbwerte, die hier zu setzen sind, werden auch im Handbuch beschrieben und sollten Ihnen geläufig sein.

Zeilen 310–360: Ausgabe der Texte. Die Hintergrundfarbe wird durch die Texte selbst bestimmt:

ungeshiftet, nicht revers:	Farbe #0 (Adr. 53281)
geshiftet, nicht revers:	Farbe #1 (Adr. 53282)
ungeshiftet, revers:	Farbe #2 (Adr. 53283)
geshiftet, revers:	Farbe #3 (Adr. 53284)

Die Vordergrundfarbe ist wie gewohnt über Steuerzeichen wählbar. Es ist noch darauf hinzuweisen, daß die geshiftet eingegebenen Zeichen ungeshiftet erscheinen, dafür aber in anderer Farbe; die »SHIFTung« dient im ECM nur der Farbkennzeichnung.
Kurz gesagt: Zugunsten zusätzlicher Farboptionen verzichtet man im ECM auf die Verfügbarkeit des gesamten Zeichensatzes und beschränkt sich auf die ersten 64 Codes, die meist auch die wichtigsten sind.

Anwendungen

Der ECM läßt sich vielfältig einsetzen, vor allem dort, wo übersichtliche Bildschirmaufbauten gefragt sind. Bei der Programmierung von Windows und komfortablen Menüs eröffnen sich dadurch weitere Möglichkeiten der grafisch ansprechenden Gestaltung, die wirklich zu sensationellen Ergebnissen führen. Das Demonstrationsprogramm war Anregung und Unterweisung zugleich; es liegt jetzt an Ihnen, den Extended Color Mode auszureizen.

Anhang: Die Masterdiskette

Die beiliegende Masterdiskette ist doppelseitig bespielt. Um Datenverlusten vorzubeugen, sollten Sie unbedingt Sicherheitskopien anfertigen und nur mit diesen arbeiten.
Die Vorderseite enthält 184 Fileeinträge, die Rückseite 106 (Trennstriche inbegriffen):
Die enorme Zahl von Dateien auf der Vorderseite wurde durch einen Trick erzielt, wie er in der C64-Geschichte einmalig ist; normalerweise sind nur 144 Einträge auf 144/8 = 18 Directory-Blöcken möglich. Durch einen besonderen Kniff konnten jedoch 5 zusätzliche Directory-Blöcke herbeigezaubert werden; diese liegen auf Spur 1 und wurden durch Manipulation der Sektorverkettung eingebunden.
Ich habe dabei jeweils den letzten verwendeten Directory-Block (18/18) aufgefüllt und, wenn er mit 8 Einträgen ausgelastet war, auf Spur 1 kopiert. Anschließend machte ich 18/18 frei und konnte weitere 8 Dateien speichern.
Da der Schreib-Lese-Kopf während des Directory-Lesens von Spur 18 auf Spur 1 wechseln muß, ertönt ein lautes Fahrgeräusch. Dies ist jedoch völlig korrekt.

```
DIRECTORY

0  [illegible]
1      "JOYCURSOR.SRC"     PRG
1      "JOYCURSOR $C000"   PRG
0      "----------------"  DEL
21     "NEUE STEUERZ.SRC"  PRG
1      "NEUE STEUERZEICH"  PRG
0      "----------------"  DEL
4      "PROGR.DIREKTMOD."  PRG
4      "PROGR.DIREKTM. 2"  PRG
3      "MEHRERE BEFEHLE"   PRG
4      "ZEILENGENERATOR"   PRG
0      "----------------"  DEL
23     "KEY-32"            PRG
5      "M-KEY-32"          PRG
0      "----------------"  DEL
1      "EZ MERGE"          PRG
1      "(EZ OLD)"          PRG
1      "EZ INITIALIZE"     PRG
1      "EZ RENUMBER"       PRG
1      "EZ TRACE"          PRG
1      "EZ DIRECTORY"      PRG
1      "EZ BEREICH-SAVE"   PRG
1      "EZ GOTO-X"         PRG
1      "EZ SPACING"        PRG
1      "EZ PROG.LIST"      PRG
1      "EZ HELP"           PRG
1      "EZ DI-AS"          PRG
1      "EZ MICROSOUND"     PRG
1      "EZ BLOCKMOVE"      PRG
1      "EZ AND/OR"         PRG
1      "EZ INPUT"          PRG
1      "EZ PSEUDO-WINDOW"  PRG
1      "EZ SCROLL-X"       PRG
1      "EZ SMOOTH-SCROLL"  PRG
1      "EZ DOWN-SCROLL"    PRG
1      "EZ DELETE-ZEILE"   PRG
1      "EZ ZEILEN-COPY"    PRG
1      "EZ PRINT-USING"    PRG
1      "EZ ZAHLFORMAT."    PRG
1      "EZ ZENTRIERUNG"    PRG
1      "EZ RECHTSBNDG"     PRG
1      "EZ DEZIMAL->BEL."  PRG
1      "EZ BEL.->DEZIMAL"  PRG
1      "EZ DIVISION"       PRG
1      "EZ FAKULTAET"      PRG
1      "EZ MATHE-GENIE"    PRG
1      "EZ ZAHLENRATEN"    PRG
1      "EZ ZAHLENRATEN2"   PRG
1      "EZ PRIMZAHLEN"     PRG
1      "EZ WOCHENTAG"      PRG
1      "EZ PI-BERECHNUNG"  PRG
```

```
1      "EZ FLOPPY-SPEEDE" PRG
1      "EZ JUSTAGE"       PRG
1      "EZ PROTECTOR"     PRG
1      "EZ TRACK-KILLER"  PRG
1      "EZ CHECKDEVICE"   PRG
1      "EZ TASTENABFRAGE" PRG
1      "EZ GRAFIK-CLR"    PRG
1      "EZ HIRES-INVERS"  PRG
1      "EZ RASTER-IRQ"    PRG
1      "EZ NEWFONT"       PRG
1      "EZ KLAVIERTON"    PRG
1      "EZ SIGNALTON"     PRG
1      "EZ PLOTTER"       PRG
1      "EZ HARDCOPY"      PRG
1      "EZ MARSMENSCH"    PRG
1      "EZ IRRGARTEN"     PRG
1      "EZ ENGLISCH"      PRG
1      "EZ ADVENTURE"     PRG
1      "EZ SCHACHUHR"     PRG
1      "EZ EZ-EDITOR"     PRG
2      "EX-LINE $C000"    PRG
1      "EZ SOFT-FLASH"    PRG
1      "EZ SOUND-FLASH"   PRG
1      "EZ STRICH-CURSOR" PRG
1      "EZ UPSIDE-DOWN"   PRG
0      "----------------" DEL
2      "4Z MASTER MIND"   PRG
1      "SOUND-BEISPIEL"   PRG
2      "3Z REFORMAT"      PRG
2      "TOKEN-FINDER"     PRG
1      "2Z SPRITE-EDITOR" PRG
1      "SPRITE-EDITOR V2" PRG
1      "3Z MINMON"        PRG
1      "2Z PYTHAGORAS"    PRG
1      "2Z FLOPPY-STATUS" PRG
1      "2Z DISK-NAME"     PRG
1      "4Z PRIMZAHLEN"    PRG
5      "23Z FROSCH"       PRG
4      "SCREEN-PONG 64"   PRG
0      "----------------" DEL
9      "MPRG IM BASIC"    PRG
1      "MERGE"            PRG
5      "STARTADRESSEN"    PRG
1      "KORREKTURZEILE"   PRG
41     "DEMO ZU MPRG"     PRG
1      "Z 1000"           PRG
1      "MAXIZEILE"        PRG
1      "RENEW"            PRG
1      "FENSTER"          PRG
1      "RESTORE X"        PRG
1      "GOTO X"           PRG
1      "TONZEICHEN"       PRG
1      "SHIFT+TON"        PRG
1      "SHIFT+WARNTON"    PRG
1      "CLR AB CURSOR"    PRG
1      "SCROLL AB X"      PRG
1      "SCROLL BIS X"     PRG
1      "1 MERGE"          PRG
1      "2 MAXIZEILE"      PRG
1      "3 RENEW"          PRG
1      "4 FENSTER"        PRG
1      "5 RESTORE X"      PRG
1      "6 GOTO X"         PRG
1      "7 TONZEICHEN"     PRG
1      "8 SHIFT+TON"      PRG
1      "9 SHIFT+WARNTON"  PRG
1      "10 CLR AB CURSOR" PRG
1      "11 SCROLL AB X"   PRG
1      "12 SCROLL BIS X"  PRG
0      "----------------" DEL
2      "KAUFM.DARSTELLG." PRG
1      "ELIMINATOR"       PRG
1      "NULL-ERGAENZUNG"  PRG
17     "INPUT (DEMO)"     PRG
6      "INPUT (KURZ)"     PRG
1      "DIRECTORY"        PRG
4      "DISK-MENUE"       PRG
9      "MISCHSORT"        PRG
7      "MISCHEN"          PRG
11     "DATUM-DEMO"       PRG
5      "INSTR-DEMO"       PRG
0      "----------------" DEL
1      "BILDSCH.LANGSAM"  PRG
1      "RE-CLR"           PRG
4      "ZEILENMARKER"     PRG
2      "PACKROUTOBJ"      PRG
1      "CLEAR-DIM"        PRG
1      "PAUSE"            PRG
1      "UNDERROMREAD"     PRG
2      "ARRAYSEEK"        PRG
1      "SCREEN-TOOL"      PRG
1      "SPRITESET"        PRG
2      "GENERAL INPUT"    PRG
1      "MENUE"            PRG
1      "DISK-TOOL"        PRG
2      "FARBEFFEKT 1"     PRG
2      "FARBEFFEKT 2"     PRG
2      "FARBEFFEKT 3"     PRG
2      "FARBEFFEKT 4"     PRG
2      "FARBEFFEKT 5"     PRG
2      "FARBEFFEKT 6"     PRG
2      "FARBEFFEKT 7"     PRG
2      "FARBEFFEKT 8"     PRG
2      "FARBEFFEKT 9"     PRG
2      "FARBEFFEKT 10"    PRG
2      "FARBEFFEKT 11"    PRG
1      "FARBEFFEKT 12"    PRG
0      "----------------" DEL
5      "OLD.SRC"          PRG
2      "OLD"              PRG
2      "£"                PRG
3      "DOS HELP $C000"   PRG
2      "TC"               PRG
58     "TOOL-CREATOR"     PRG
14     "PARADOXON-BASIC"  PRG
4      "UEBERS. LISTING"  PRG
11     "CROSS-REF 64"     PRG
8      "BASIC-PACKER"     PRG
5      "LMS"              PRG
4      "LMS.PHA"          PRG
23     "LMS.BAS"          PRG
0      "----------------" DEL
7      "MENUE-BSP. 1"     PRG
11     "MENUE-BSP. 2"     PRG
7      "MENUE-BSP. 3"     PRG
14     "MENUE-BSP. 4"     PRG
5      "MENUE-BSP. 5"     PRG
6      "MENUE-BSP. 6"     PRG
5      "MENUE-BSP. 7"     PRG
7      "MENUE-BSP. 8"     PRG
6      "MENUE-BSP. 9"     PRG
6      "MENUE-BSP. 10"    PRG
5      "MENUE-BSP. 11 BA" PRG
26     "C/MENUE-BSP. 11"  PRG
0 BLOCKS FREE.
```

```
0 [illegible]
6      "WINDOW-BSP. 1"      PRG
9      "WINDOW-BSP. 2"      PRG
8      "WINDOW-BSP. 3"      PRG
26     "WINDOW-BSP. 3CMP"   PRG
38     "WINDOW-BSP. 4SRC"   PRG
4      "WINDOW-BSP. 4OBJ"   PRG
0      "----------------"   DEL
0      "----------------"   DEL
6      "MASKENGEN.49152"    PRG
0      "----------------"   DEL
0      "----------------"   DEL
25     "HYPRA-ASS"          PRG
4      "LISTING 10.1 (B)"   PRG
3      "LISTING 10.2 (S)"   PRG
1      "LISTING 10.2 (O)"   PRG
3      "LISTING 10.3 (S)"   PRG
1      "LISTING 10.3 (O)"   PRG
3      "LISTING 10.4 (S)"   PRG
1      "LISTING 10.4 (O)"   PRG
4      "LISTING 10.5 (S)"   PRG
1      "LISTING 10.5 (O)"   PRG
5      "LISTING 10.6 (S)"   PRG
1      "LISTING 10.6 (O)"   PRG
5      "LISTING 10.7 (S)"   PRG
1      "LISTING 10.7 (O)"   PRG
3      "LISTING 10.8 (S)"   PRG
1      "LISTING 10.8 (O)"   PRG
4      "LISTING 10.9 (S)"   PRG
1      "LISTING 10.9 (O)"   PRG
8      "LISTING 10.10(S)"   PRG
1      "LISTING 10.10(O)"   PRG
25     "LISTING 10.11(S)"   PRG
3      "LISTING 10.11(O)"   PRG
1      "LISTING 10.12"      PRG
1      "LISTING 10.13"      PRG
1      "LISTING 10.14"      PRG
1      "LISTING 10.15"      PRG
1      "LISTING 10.16"      PRG
1      "LISTING 10.17"      PRG
1      "LISTING 10.18"      PRG
1      "LISTING 10.19"      PRG
1      "LISTING 10.20"      PRG
1      "LISTING 10.21"      PRG
1      "LISTING 10.22"      PRG
11     "LISTING 10.23(S)"   PRG
1      "LISTING 10.24"      PRG
1      "LISTING 10.25"      PRG
1      "LISTING 10.26"      PRG
1      "LISTING 10.27"      PRG
1      "LISTING 10.28"      PRG
2      "LISTING 10.29(S)"   PRG
28     "LISTING 10.30"      PRG
3      "LISTING 10.31"      PRG
39     "LISTING 10.32(L)"   PRG
6      "LISTING 10.32(O)"   PRG
0      "----------------"   DEL
0      "----------------"   DEL
2      "!"                  PRG
0      "----------------"   DEL
3      "FLOPPYLISTER"       PRG
0      "----------------"   DEL
2      "AUTOSTART"          PRG
0      "----------------"   DEL
3      "FILE-COMPRESSOR"    PRG
7      "FILE-COMPACTOR"     PRG
0      "----------------"   DEL
6      "DISK-OPTIM. II"     PRG
0      "----------------"   DEL
42     "DISC-WIZARD"        PRG
0      "----------------"   DEL
23     "REFORMAT"           PRG
0      "----------------"   DEL
12     "SUPER COPY"         PRG
0      "----------------"   DEL
2      "PRODISC"            PRG
1      "PRORETTER"          PRG
32     "CCTHRG"             PRG
5      "CCTSPR"             PRG
4      "CCTM1A"             PRG
4      "CCTM1B"             PRG
7      "CCTM2"              PRG
1      "CCTM3"              PRG
1      "CCTM4"              PRG
17     "CCTM5"              PRG
0      "----------------"   DEL
5      "FAST FORMAT"        PRG
1      "HILFSPROGRAMM FF"   PRG
0      "----------------"   DEL
19     "DISK FUELLER"       PRG
0      "----------------"   DEL
15     "DISKTESTER"         PRG
0      "----------------"   DEL
15     "HYPRA-COPY"         PRG
0      "----------------"   DEL
20     "MASTER-COPY V1.7"   PRG
0      "----------------"   DEL
16     "COPY+"              PRG
0      "----------------"   DEL
0      "----------------"   DEL
1      "SPINDIZZY TRAIN."   PRG
15     "SPRITEKILL"         PRG
1      "ADVENTURE-LISTER"   PRG
0      "----------------"   DEL
0      "----------------"   DEL
5      "BSP/EXT.COL.MODE"   PRG
0      "----------------"   DEL
71 BLOCKS FREE.
```



Stichwortverzeichnis